

MISSION SERVICES PROGRAM

Space Network (SN) Web Services Interface (SWSI) System Design Specification

DRAFT

October 2000



National Aeronautics and
Space Administration

Goddard Space Flight Center
Greenbelt, Maryland

Space Network (SN) Web Services Interface (SWSI) System Design Specification

October 2000

Prepared Under Contract NAS 9-98100

Prepared by:

Harshna Sampat
Computer Sciences Corporation

Date

Prepared by:

Gerald Klitsch
Computer Sciences Corporation

Date

Prepared by:

Thomas E. Sardella
SWSI Product Manager

Date

Approved by:

Date

Goddard Space Flight Center
Greenbelt, Maryland

Preface

This System Design Specification describes the detailed design for the Space Network (SN) Web Services Interface (SWSI) in support of operations of the National Aeronautics and Space Administration (NASA) Goddard Space Flight Center (GSFC) Network Control Center (NCC) and of the Demand Access System (DAS) located at the White Sands Complex (WSC).

Changes to this document shall be made by Documentation Change Notice (DCN) or by complete revision.

Questions concerning this document or proposed changes should be addressed to:

National Aeronautics and Space Administration (NASA)
Goddard Space Flight Center
Technology and Mission Upgrades Project
Code 453
Greenbelt, Maryland 20771

Abstract

The primary function of the Space Network (SN) Web Services Interface (SWSI) is to provide a standards-based cross-platform customer interface for performing Tracking and Data Relay Satellite (TDRS) and Demand Access System (DAS) scheduling and real-time service monitoring and control. A secure interface will be provided to allow these functions to be performed either from the NASA Integrated Services Network (NISN) Internet Protocol (IP) Operational Network (IONET) or from the Internet.

This System Design Specification presents the detailed design for the SWSI.

Keywords: *SWSI, NCCDS, SN, DAS, DASCON*

Change Information Page

List of Effective Pages			
Page Number	Issue	Page Number	Issue

Document History			
Document Number	Status/Issue	Publication Date	CCR Number

DCN Control Sheet

[illegible]

CONTENTS

PREFACE	III
ABSTRACT	IV
CHANGE INFORMATION PAGE	V
DCN CONTROL SHEET	VI
Figures	xi
SECTION 1. INTRODUCTION	1
1.1 Purpose	1
1.2 Background.....	1
1.3 Scope.....	2
1.4 Document Organization.....	2
1.5 Applicable Documents	3
SECTION 2. DESIGN OVERVIEW	1
2.1 Overview.....	1
2.2 System Environment.....	4
Network Control Center (NCC) Data System (NCCDS) Operations	4
NCCDS TDRSS Unscheduled Time (TUT) Server.....	5
NISN Secure Gateway	6
2.3 SWSI Platform.....	6
2.4 Software Overview.....	6
2.5 Development Approach.....	8
2.5.1 Standards/Methodology.....	8
2.5.2 Development Environment	9
2.5.3 Development Tools and Software.....	10
SECTION 3. CLIENT DESIGN	1
3.1 Overview.....	1

3.2	Use Cases.....	1
3.3	Client User Interface.....	5
3.3.1	Main Panel.....	5
	SWSI.....	6
3.3.2	Login Panel.....	6
3.3.3	Schedule Request Panels	8
3.3.3.1	Schedule Add Request (SAR) Panel.....	9
3.3.3.2	Schedule Delete Request (SDR) Panel.....	12
3.3.3.3	Alternate Schedule Add Request (ASAR) Panel.....	13
3.3.3.4	Replace Request (RR) Panel.....	13
3.3.3.5	Wait List Request (WLR) Panel.....	14
3.3.3.6	Resource Allocation Request (RAR) Panel.....	14
3.3.3.6	Resource Allocation Request (RAR) Panel.....	15
3.3.3.7	Resource Allocation Modification Request (RAMR) Panel.....	16
3.3.3.8	DAS Playback Modification Request Panel.....	17
3.3.4	View Schedule Requests Panel.....	18
3.3.5	View Active Schedule Panel.....	21
3.3.5.1	View Service Display Panel.....	23
3.3.6	Alert Panel.....	25
3.3.7	GCMR Panel.....	27
3.3.7.1	GCMR Menu Panel for NCC Services.....	27
3.3.7.2	GCMR Menu Panel for DAS Services.....	28
3.3.7.3	Service Reconfiguration Panel.....	29
3.3.8	UPD.....	30
3.3.8.1	UPD Summary Panel.....	30
3.3.8.2	UPD Detail Panels	32
3.3.8.3	UPD Processing	34
3.3.9	TSW & State Vectors.....	35
3.3.10	ServiceParmWindow.....	37
3.3.11	SSC Editing Panel.....	41
3.3.12	DAS Resource Availability.....	42
3.3.13	DAS Playback Planning.....	44
3.4	Data Manager.....	45
3.5	Logging	48
SECTION 4. APPLICATION SERVER DESIGN		1
4.1	Overview.....	1
4.2	Detailed Design.....	3

4.3	Data Interfaces.....	4
4.4	Logging	4
SECTION 5. ISOLATOR DESIGN		1
5.1	Overview.....	1
5.2	Isolator Main Task (MainTask):.....	4
5.3	Application Server Interface (ServInterface):	4
5.3.1	TP1 Port.....	4
5.3.1.1	SNIF TP1 Messages.....	5
5.3.1.2	SDIF TP1 Messages	6
5.3.1.3	Users TP1 Messages.....	9
5.3.2	TP2 Port.....	11
5.3.3	TP3 Port.....	12
5.4	Database Interface (DbInterface):	14
5.5	SNIF Interface (SnifInterface):.....	14
5.6	SDIF Interface (SdifInterface):.....	15
5.7	Logging	15
SECTION 6. SWSI-NCCDS INTERFACE DESIGN		1
6.1	Overview.....	1
6.2	Operating Environment.....	1
6.3	Detailed Design.....	1
6.3.1	Read Isolator Messages.....	3
6.3.2	NCCDS Interface	3
6.3.2.1	Manage NCCDS Communications	5
6.3.2.1.1	Schedule Request.....	5
6.3.2.1.2	Schedule Result Message	5
6.3.2.1.3	User Schedule Message	6
6.3.2.1.4	TDRS Scheduling Window Message.....	6
6.3.2.1.5	State Vector.....	6
6.3.2.1.6	User Performance Data.....	6
6.3.2.1.7	Acquisition Failure Notification.....	6
6.3.2.1.8	Return Channel Time Delay Message.....	6
6.3.2.1.9	Time Transfer Message	6
6.3.2.1.10	Ground Control Message Request.....	7

6.3.2.1.11	Ground Control Message Status and Disposition.....	7
6.3.2.2	Send Schedule Request.....	7
6.3.2.3	Receive Schedule Result	7
6.3.2.4	Send TDRSS Scheduling Window.....	7
6.3.2.5	Send State Vector	8
6.3.2.6	Receive pmdata	8
6.3.2.7	Send Ground Control Message Request.....	8
6.3.2.8	Receive Ground Control Message Status and Disposition.....	8
6.3.2	Logging and Delogging	8
7.	SWSI-DAS INTERFACE DESIGN.....	1
7.1	Overview.....	1
7.2	SDIF Functionality	2
7.2.1	Isolator to DASCON Interface	2
7.2.1.1	Detailed design.....	2
7.2.2	DASCON to Isolator Interface	3
7.2.2.1	Detailed Design.....	3
7.2.3	Retransmission Thread.....	3
7.3	Database Interface.....	7
7.4	Support for test and operational modes	7
SECTION 8.	DATABASE DESIGN	1
8.1	Design Principles and Guidelines	1
8.2	The SWSI Database Design.....	1
8.2.1	Overview.....	1
8.2.2	Stored Procedures	7
8.2.2.1	SWSI_ActiveSchedule_pkg.....	7
8.2.2.2	SWSI_ScheduleRequest_pkg.....	8
8.2.2.3	SWSI_ScheduleResponse_pkg.....	9
8.2.2.4	SWSI_GCMR_pkg.....	10
8.2.2.5	SWSI_SSCEdit_pkg.....	11
8.3	Database Configuration	11
8.4	Database Maintenance.....	11
8.4.1	Synchronization with NCCDS.....	11
8.4.2	Purging	11
8.4.3	Backup and Recovery	11

8.5	Operational Considerations	12
SECTION 9. TUT SERVER		1
SECTION 10. SECURITY.....		1
10.1	Security Requirements	1
10.2	Security Model.....	1
10.3	Security Features	2
APPENDIX A – COMMON CLASSES		1
APPENDIX B - TRACEABILITY		1
APPENDIX C - ISOLATOR-SNIF INTERFACE		1
APPENDIX D – ISOLATOR-SDIF INTERFACE		1
TBS		1
APPENDIX E – ISOLATOR OBJECT TYPES DESCRIPTION		1
APPENDIX F – SWSI DATABASE TABLES		1
Table Name		1
Type of Table.....		1
Description		1
ABBREVIATIONS AND ACRONYMS		1

Figures

Figure 2-1	High Level SWSI Architecture.....	4
Figure 2-2	High Level SWSI Dataflow Diagram.....	7
Figure 2-3	SWSI Development Directory Structure	10
Figure 3-1	SWSI Use-Case Diagram.....	2
Figure 3-2	SWSI Extended Use-Case Diagrams.....	4
Figure 3-3	Main Control Panel.....	5
Figure 3-4	Main Control Panel Menu Options	6

Figure 3-5	Connection Parameters Panel.....	7
Figure 3-6	Schedule Add Request Panel.....	9
Figure 3-7	Example of respecifiable panel alone for KaSAR service.....	11
Figure 3-8	Edit Service Flexibility Parameters Panel.....	12
Figure 3-9	Delete Request Panel.....	13
Figure 3-10	Wait List Request Panel.....	14
Figure 3-11	DAS Resource Allocation Request Panel.....	16
Figure 3-12	Resource Allocation Modification Request Panel Constructors.....	16
Figure 3-13	DAS Resource Allocation Modification Request Panel.....	17
Figure 3-14	DAS Playback Modification Request.....	18
Figure 3-15	Schedule Requests Panel.....	19
Figure 3-16	Class Diagrams for Schedule Requests.....	20
Figure 3-17	Active Schedule Panel.....	21
Figure 3-18	Class Diagrams of Active Schedule	22
Figure 3-19	Service Display Panel.....	23
Figure 3-19a	DAS TDRS Handovers Panel.....	24
Figure 3-20	Class Diagrams of Service Display.....	25
Figure 3-21	Alert Panel.....	26
Figure 3-22	DAS GCM Menu Panel.....	28
Figure 3-23	UPD Summary Panel.....	31
Figure 3-24	UPD Detail Panel.....	33
Figure 3-25	State Vector Input Panel.....	37
Figure 3-26	Class Diagram of ParameterizedRequest Interface.....	38
Figure 3-27	Class Diagram of ServiceParmWindow.....	39
Figure 3-28	Class Diagram of ServiceBean Interface	40
Figure 3-29	DAS SSC Editing Panel.....	41
Figure 3-30	DAS Resource Availability Request Panel.....	42
Figure 3-31	DAS Availability Panel.....	43
Figure 3-32	DAS Playback Planning Panel.....	44
Figure 3-33	DAS Playback Availability Report.....	45
Figure 3-34	Data Request Procedure.....	46
Figure 3-35	Event Class and Listener Interface.....	47
Figure 3-36	DataValue Class Diagram.....	47
Figure 3-37	DataManager Class Diagram.....	48
Figure 4-1	SWSI Server Design	2
Figure 5-1	Communication Flow of SWSI elements.....	1
Figure 5-2	Isolator Context Diagram.....	3
Figure 5-3	Isolator Main Threads.....	4
Figure 5-4	Data flow of TP1 Messages bound to SNIF/NCC	6
Figure 5-5	Flow of TP1 DAS Messages that get stored in SWSI database.....	8
Figure 5-6	Flow of DAS Messages not stored in SWSI database	9
Figure 5-7	Flow of TP1 Common User Request Messages.....	10

Figure 5-8	TP2 Data flow Messages.....	12
Figure 5-9	TP3 Data Flow of the Alerts Messages.....	13
Figure 6-1	SNIF Context Diagram.....	2
Figure 6-2	SNIF Level 0 Data Flow Diagram.....	3
Figure 6-3	NCCDS Interface Data Flow Diagram.....	4
Figure 6-4	ANCC Interface Data Flow Diagram.....	9
Figure 7-1	SDIF Context Diagram.....	1
Figure 7-2	Control Logic Overview.....	4
Figure 7-3	Detailed Control Logic.....	5
Figure 7-4	Retransmission Control Logic	6
Figure 8-1	SWSI Database Schema (part 1 of 3).....	3
Figure 8-2	SWSI Database Schema (part 2 of 3).....	4
Figure 8-3	SWSI Database Schema (part 3 of 3).....	5
Figure 8-4	SWSI Database Table Views.....	6
Figure A-1	Common Class Diagram 1	5
Figure A-2	Common Class Diagram 2	6
Figure A-3	Common Class Diagram 3	7
Figure A-4	Common Class Diagram 4	8
Figure A-5	Common Class Diagram 5	9
Figure A-6	SAR Common Class Diagram.....	11
Figure A-7	SDR Common Class Diagram.....	11
Figure A-8	ASAR Common Class Diagram.....	12
Figure A-9	RR Common Class Diagram.....	13
Figure A-10	WLR Common Class Diagram.....	14
Figure A-11	MnemonicRequest Common Class Diagram.....	15
Figure A-12	DAS Requests Common Class Diagram.....	16
Figure A-13	DAS Availability Common Class Diagrams	17
Figure A-14	SSC Support Common Class Diagrams	18
Figure A-15	DAS GCMR Support Common Class Diagrams	19

Section 1. Introduction

1.1 Purpose

This document describes in detail the hardware and software design for the Space Network (SN) Web Services Interface (SWSI). The primary goal of SWSI is to provide a standards-based customer interface for performing Tracking and Data Relay Satellite (TDRS) and Demand Access System (DAS) scheduling and real-time service monitoring and control. The intent of the SWSI is not to replace existing scheduling and real-time systems for all SN customers. It is rather to provide a simple low-cost interface option, especially for suborbital and infrequent SN customers. SWSI does however provide the primary customer interface for all DAS customers.

1.2 Background

The interface between a customer Mission Operations Center (MOC) and the Network Control Center Data System (NCCDS) consists of formatted messages exchanged electronically using either Nascom 4800 Bit Block (BB) protocol or Transmission Control Protocol (TCP). This interface is described in detail in the NCCDS/MOC Interface Control Document (ICD). New SN customers have traditionally been provided with a limited number of options for implementing this interface. A full-featured SN scheduling tool is provided by the User Planning System (UPS), which runs on a Hewlett-Packard (HP) Unix host. New customers desiring to use UPS for scheduling must either purchase their own system or interface with an institutional UPS located within the Multisatellite Operations Control Center (MSOCC). A NASA Integrated Services Network (NISN) Closed Internet Protocol (IP) Operational Network (IONET) connection is required for the latter option.

No standard option exists to provide a real-time (reconfiguration and performance data monitoring) interface. All SN customers have been required to implement their own systems at considerable cost.

Prospective SN customers have brought to light the need for a simple, standard, readily available interface to the NCCDS. In response to this need, NASA funded an in-house project to determine the feasibility of such a tool. This project resulted in a prototype of a web-based cross-platform customer interface to the NCCDS, called the SN Web Services Interface (SWSI). Prototyping and proof of concept work was completed and has been used to provide support to the Long Duration Balloon Project (LDBP).

The final operational SWSI is a follow-on to the prototype effort and will provide improvements in the form of a Java-based Graphical User Interface (GUI) and better management of user schedule information. Using the SWSI, SN customers will be able to perform scheduling, real-time functions, and state vector storage for only the cost of a desktop computer or workstation. A web browser and a Java virtual machine, both of which are freely available, will also be required. The SWSI is designed to be accessed from the NISN Closed IONET or Open IONET. NISN's Open IONET allows access

from the NASA Science Internet and the public Internet, thus allowing cooperation with NASA's university, enterprise, and inter/intra-agency partners.

In addition to providing this interface to the NCCDS for legacy SN services, the SWSI will provide the customer interface for scheduling the newer DAS Multiple Access Return (MAR) services. The advantage to the DAS Project is that SWSI already provides the infrastructure needed by DAS to provide similar customer interface capabilities. Adding a DAS interface to SWSI will spare the DAS Project the expense of duplicating those facilities and will provide a single integrated application for customers to perform both legacy SN and DAS scheduling, monitoring, and control.

1.3 Scope

This document describes the proposed design of the SWSI, including the hardware and software architectures, subsystem designs, and software module definitions. This is the primary document used in describing the design and forms the basis for implementation of the system.

1.4 Document Organization

This document is organized into nine sections and four appendices. Following the Introduction (Section 1), this document presents the SWSI Detailed Design in the following order:

- Design Overview (Section 2)
- Client Design (Section 3)
- Application Server Design (Section 4)
- Isolator Design (Section 5)
- SWSI-NCCDS Interface Design (Section 6)
- SWSI-DAS Interface Design (Section 7)
- Database Design (Section 8)
- TUT Server (Section 9)
- Security (Section 10)
- Common Classes (Appendix A)
- Traceability (Appendix B)
- Isolator-SNIF Interface (Appendix C)
- Isolator-SDIF Interface (Appendix D)
- Isolator Object Types Description (Appendix E)
- SWSI Database Tables (Appendix F)

- Abbreviations and Acronyms

1.5 Applicable Documents

1. *Network Control Center Data System (NCCDS) System Requirements, 1998, 530-SRD-NCCDS/1998*
2. *Interface Control Document Between the Network Control Center Data System and Mission Operations Center, 530-ICD-NCCDS/MOC*
3. *Demand Access System (DAS) Systems Requirements Document, 451-SRD-DAS*
4. *Interface Control Document Between the Demand Access System and the Space Network Web Services Interface, 451-ICD-DAS/SWSI*
5. *NCCDS Protocol Gateway Operator's Guide Release 98.1, 451-NPGUG/NCC98*
6. *High Availability User's Guide Release 98.1, 451-HAUG/NCC98*
7. *NCC Central Delogger (NCD) Operations Concept, 530-NCD-NCC98, May 1997*
8. *NCCDS Specification for World Wide Web Server for TDRSS Unscheduled Time and Nascom Information (Draft), October 1996*
9. *Java-based Spacecraft Web Interface to Telemetry & Command Handling (Jswitch) System Design, August 1999*
10. *Java-based Spacecraft Web Interface to Telemetry & Command Handling (Jswitch) User's Guide, April 2000*
11. *NASA Procedures and Guidelines (NPG) 2810.1, Security of Information Technology, August 1999*
12. *NASA Policy Directive (NPD) 2810.1, NASA Policy for Security of Information Technology, October 1998*
13. *Security Plan for the Network Control Center, NCC 98, 451-SP-NCC/1998, April 1998*
14. *Security Plan for Space Network Web Services Interface, 452-SP-SWSI, May 10, 2000*
15. *NASA GSFC Data Systems Technology Java™ Style Guide, July 1997*
16. *IP Operational Network (IONet) Security Plan, 290-003, September 1999*

Section 2. Design Overview

2.1 Overview

The primary function of the Space Network (SN) Web Services Interface (SWSI) is to provide a Java-based web interface to the NCCDS and to DAS to perform customer scheduling, real-time service monitoring and control, and state vector storage. The SWSI performs the following major functions:

- Support all full support customer messages as defined in the NCCDS/MOC ICD.
- Support all messages as defined in the DAS/SWSI ICD.
- Allow a customer to submit all schedule request messages as defined in the ICDs.
- Maintain a database of customer Service Specification Codes (SSCs) that matches the NCCDS database to assist the customer in generating schedule requests.
- Allow for customer maintenance of DAS-specific SSCs.
- Provide for customer scheduling of DAS playback events.
- Maintain an active schedule file derived from Schedule Result Messages (SRMs) and User Schedule Messages (USMs) received from the NCCDS.
- Provide an Active Schedule display consisting of events for both NCCDS and DAS scheduled services.
- Allow a customer to generate Ground Control Message Requests (GCMRs) and display results received from the NCCDS.
- Allow a customer to generate DAS Service Reconfiguration Messages and display results received from DAS.
- For each active NCCDS event, maintain a list of current parameter settings that reflects initial values and any parameters changed in response to User Reconfiguration Request messages.
- Provide for monitoring of NCCDS and DAS User Performance Data (UPD) in user-configurable displays.
- Store Return Channel Time Delay Messages (RCTDM) and Time Transfer Messages (TTM) received from the NCCDS in binary files on the customer workstation for later processing by customer applications.
- Generate Type 8 (stationary) state vectors based on customer entry of latitude, longitude, and altitude and forward them to NCCDS and/or DAS, depending on which system(s) is used to support that spacecraft.

- Allow a user to import state vectors and forward them to NCCDS and/or DAS, depending on which system(s) is used to support that spacecraft.
- Provide simultaneous access to both the operational NCCDS and the Auxiliary NCC (ANCC) for performing Engineering Interface (EIF) testing.
- Allow access from NASA Integrated Services Network (NISN) Closed IONET, Open IONET, and Internet.
- Provide for secure message exchange using encryption
- Log all formatted messages exchanged with the NCCDS and DAS, as well as significant events and errors. Provide a delogging capability to allow an operator to view logs.
- Provide a High Availability (HA) configuration to adhere to existing NCCDS Reliability/Maintainability/Availability (RMA) requirements.
- Provide customer access to Tracking and Data Relay Satellite System (TDRSS) Unscheduled Time (TUT) information from the Open IONET and Internet.

SWSI will not provide any of the shuttle specific support services.

A block diagram showing the high level SWSI architecture is given in Figure 2-1. The architecture is based on the Java-based Spacecraft Web Interface to Telemetry & Command Handling (Jswitch). Jswitch performs a similar function to SWSI in that it provides a standards-based secure remote user interface across open and closed networks to a MOC. SWSI will use a Jswitch backbone with a new Isolator, an extensively modified Client subsystem, and an enhanced Application Server.

The SWSI consist of three components and seven subsystems. The SWSI components are Client, Open Server, and Backend Server and the subsystems are Client, Application Server, Isolator, SWSI-NCCDS Interface (SNIF), SWSI-DAS Interface (SDIF), database, and Open TUT Server. The Client component is the user's desktop, which can be any desktop that supports Java Virtual Machine (JVM) 1.2. The Client subsystem is executed on the Client's component and its main function is to allow remote users to schedule Space Network resources with NCCDS and to provide Graphical User Interface (GUI) to monitor status of the scheduled resources.

The second component is the mid-tier server called Open SWSI Server. It hosts the SWSI Application subsystem, Open TUT Server subsystem, and other COTS packages; i.e. web server, and security tools e.g. IP filtering, tcp_wrapper). It is a SUN Ultra 2 sparc workstation connected on the Open IONet. The main function of the Application Server subsystem is to keep track of the user requests and provide the requested information to the Client subsystem.

There is a NASA Integrated Services Network (NISN) Secure Gateway between the Open Server and the Backend Server. The instance of the SWSI Application Server running on the Open Server component is for the Open IONet and Internet users. It also acts a proxy server for the Client component and minimizes the "holes" required on the NISN Secure Gateway to support SWSI users.

The third component is the Closed Server. It is a SUN Ultra 2 sparc workstation connected on the Closed IONet. It hosts SNIF, SDIF, two instances of the Isolator subsystem and an instance of the Application Server subsystem. It is also used as the SWSI database server. The instance of the Application Server subsystem is for the closed IONet users. One instance of the Isolator connects with the Application Server running on the Open Server and the other connects with the Application Server running on the Backend Server.

The Application Server subsystem communicates with the Client and Isolator subsystems via secure SSL connections. The Isolator communicates with SNIF and SDIF, which will handle all communications with the NCCDS and the DAS Controller (DASCON).

The Open Server component and the Backend Server component consists of two physical hosts for redundancy. The Backend Server contains SWSI data including user's information in the shared database storage. The Backend Server is responsible for all communications with the NCCDS and with DASCON.

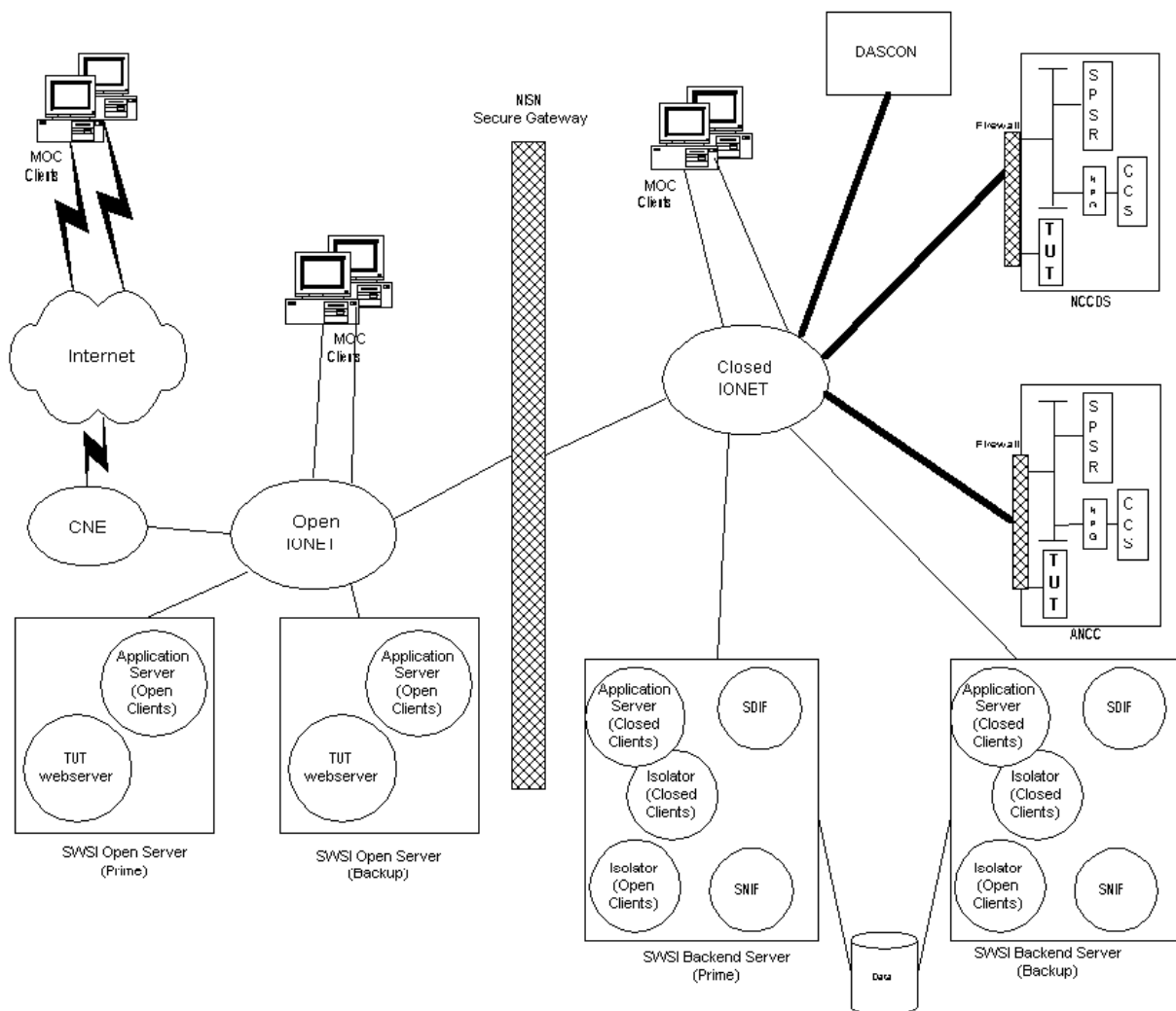


Figure 2-1 High Level SWSI Architecture

2.2 System Environment

This section describes the environment in which the SWSI operates and briefly discusses the interactions with external systems with which the SWSI interfaces.

Network Control Center (NCC) Data System (NCCDS) Operations

The NCC serves as the central control facility of the Spaceflight Tracking and Data Network (STDN), which consists of the Space Network (SN) and Ground Network (GN). The SN includes the Tracking and Data Relay Satellites (TDRSs) and two ground terminals, the White Sands Ground Terminal (WSGT) and the Second TDRSS Ground Terminal (STGT). The NCC schedules, controls, and ensures the reliability of the SN. The SWSI communicates with the operational NCCDS on behalf of SWSI customers through implementation of the NCCDS/MOC Interface Control Document (ICD)

protocol. All communications use Transmission Control Protocol (TCP) and are limited to those messages designated for full support customers.

Auxiliary Network Control Center (ANCC)

The ANCC serves primarily as a test facility for testing new NCCDS software releases and for performing Engineering Interface (EIF) tests with customer MOCs. It also functions as a backup facility to the operational NCC should facility evacuation be required. The SWSI will interface to the ANCC to allow SWSI customers to perform interface testing.

Service Planning Segment Replacement (SPSR)

The SPSR is the primary NCCDS subsystem used for performing SN service planning. SPSR receives and validates customer service requests, generates and maintains the schedule, and disseminates the schedule to the appropriate SN elements and customers. The SPSR also receives acquisition data from the Flight Dynamics Facility (FDF) and SN customers, stores the data, and disseminates acquisition data to WSGT and STGT. The SWSI maintains TCP connections with SPSR for performing scheduling and vector storage on behalf of each SWSI customer.

Communications and Control Segment (CCS)

The CCS is the primary NCCDS subsystem used for performing SN service control and service assurance. Customers are able to perform real-time reconfiguration of an ongoing service through the use of Ground Control Message Requests (GCMRs). CCS is used to monitor the performance of active events and passes this information to customers in the form of User Performance Data (UPD) messages.

NCCDS Protocol Gateway (NPG)

The NPG performs message protocol translation between legacy entities that communicate in 4800 BBs and newer entities that use TCP messages. Since CCS communicates using 4800 BB protocol and the SWSI communicates using TCP, the SWSI will establish real-time connections with the NPG, using the NPG as a TCP proxy for the CCS.

NCCDS TDRSS Unscheduled Time (TUT) Server

The TUT World Wide Web (WWW) Server provides information about unscheduled TDRS resources. It consists of start and stop times of unscheduled use of the Single Access (SA), Multiple Access Forward (MAF), and S-band Multiple Access Forward (SMAF) antennas, and Multiple Access Return (MAR) and S-band Multiple Access Return (SMAR) links for each TDRS. This data is essentially the unused time in the schedule, with a few adjustments due to flexible events with flexible start and stop times and/or flexible resources. The NCCDS TUT Server provides this service only to customers located on the Closed IONET.

Demand Access System (DAS)

The DAS expands the existing TDRSS Multiple Access Return (MAR) capabilities by building upon the Third Generation Multiple Access Beamforming Subsystem (TGBFS). The existing TDRSs provide pre-scheduled communication service to customers by using ground-based electronics to process signals emanating from customers that are relayed by the TDRS on-board phased array antenna

systems. The TGBFS expands the capability of the TDRSs MAR system and will allow service to be provided on a demand basis rather than on a pre-scheduled basis.

DAS Controller (DASCON)

DASCON is responsible for scheduling and controlling all DAS-related hardware at the White Sands Complex (WSC). The SWSI communicates with the DASCON on behalf of SWSI customers through implementation of the DAS/SWSI Interface Control Document (ICD) protocol. All communications use Transmission Control Protocol (TCP).

NISN Secure Gateway

The NISN Secure Gateway is a rule-based firewall used to prevent penetration of hosts on the Closed IONET from less secure networks. A small number of rules is used to allow connection between the Open Server and the Backend Server components. All message traffic will be channeled through this path using encrypted Secure Socket Layer (SSL) connections. The rule set will remain static, meaning that Secure Gateway changes will not be required in response to SWSI customers being added or removed.

Mission Operations Centers

All the SN customers are responsible for the design, development, maintenance, and operation of their own Mission Operations Centers (MOCs). SWSI users are required to provide workstations (desktops) within their MOCs to connect to and operate with the SWSI servers located on either the open or closed networks. Minimum workstation requirements include a web browser and a Java Virtual Machine (JVM) version 1.2. A Java Client subsystem will be provided by SWSI to the user to allow him/her to connect to the Application Server using SSL-encrypted connections. The Client subsystem will also provide the user-interface and all displays required by the customer to perform SWSI-related functions.

2.3 SWSI Platform

The Application Servers are hosted on Sun Ultra 2 Sparc computers, each initially configured with a 9 Mbyte hard disk and 128 Mbyte RAM. Quad network interface cards provide redundant heartbeat interfaces for a High Availability (HA) configuration. The operating system level is Solaris 2.7.

2.4 Software Overview

There are seven subsystems of the SWSI as shown in the high level data-flow diagram in Figure 2-2:

- Client
- Application Server
- Isolator
- SWSI-NCCDS Interface (SNIF)
- SWSI-DAS Interface (SDIF)
- Database
- Open TUT Server

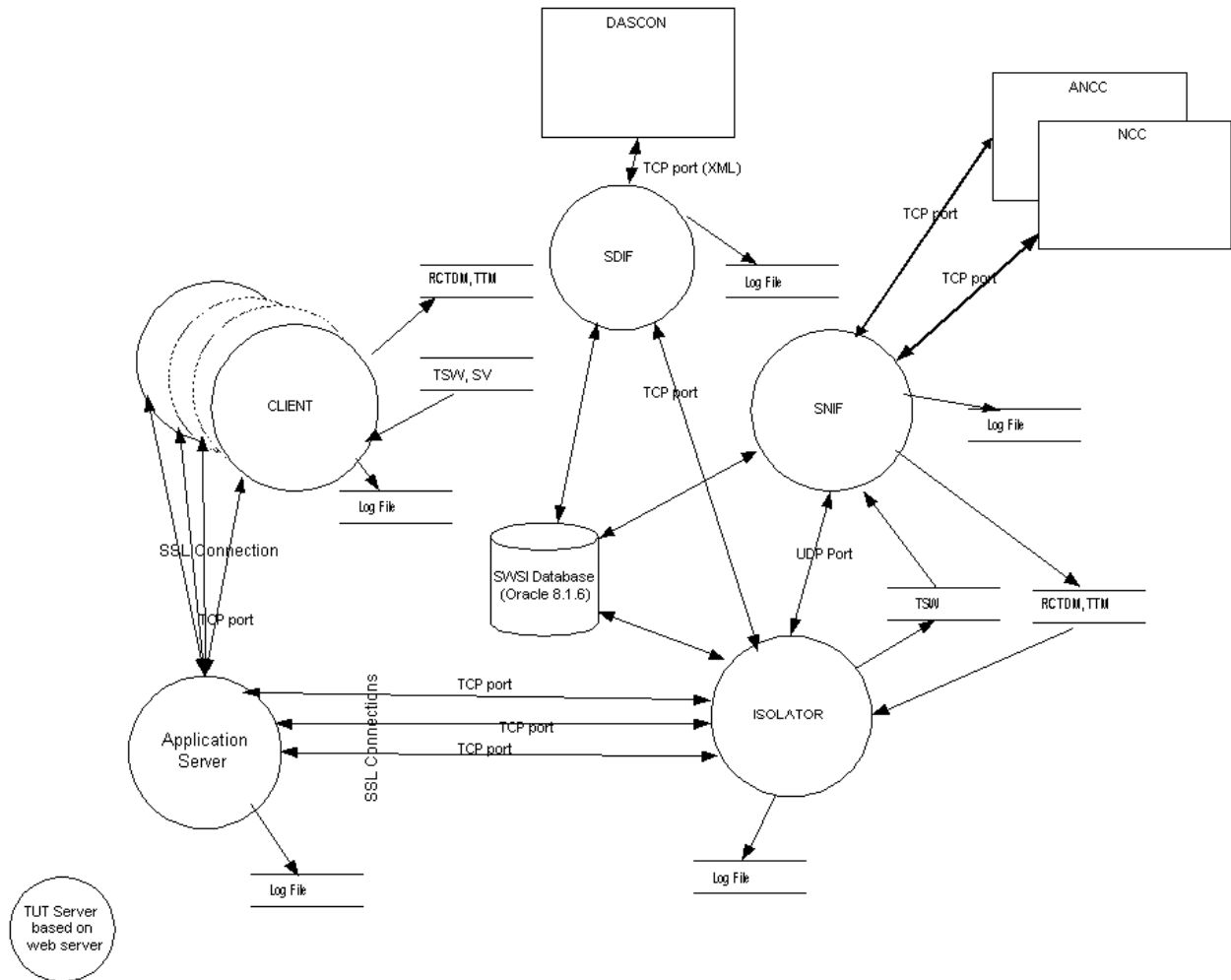


Figure 2-2 High Level SWSI Dataflow Diagram

The SWSI design relies primarily on Java supplied Application Programming Interfaces (APIs) to provide the data encapsulation, transport, and routing mechanisms. This "Pure Java" backbone allows for the widest possible use among platforms (platform independence). The use of Java allows the design and construction of platform independent user-interfaces using standard GUI components such as labels, text-fields, buttons lists, and drop-down boxes. It also ensures safe network delivery, streamlines software development and deployment, and reduces product development life cycles. Furthermore, use of the JavaBean component-based architecture allows for expandability of the system. JavaBeans will be used as the GUI components in the Client application.

- 1) SWSI is a multi-tiered architecture using a Java application for the front end. This front end is referred to as the Client subsystem and can run on any computer, which supports Java Virtual Machine (JVM) for JDK version 1.2. The instance of the Application Server that runs on the Open Server (Sun Ultra 2 Sparc computer running Solaris 2.7) acts a proxy server for the Client subsystem. The Application Server listens on one port for the Clients to establish a secure connection with the Application Server. The Application Server listens on three ports for the Isolator to establish secure connections. See Table xx for details of how the ports are used.

The Isolator communicates with the SNIF using User Datagram Protocol (UDP) and with the SDIF using Transmission Control Protocol (TCP). The types of messages between the Isolator and the SNIF and SDIF are Key Info, File Info, Alerts, and actual NCCDS and DAS messages.

The SNIF will establish connections and communicate with the NCCDS using the protocols and message formats as defined in the NCCDS/MOC ICD. The SDIF will establish and communicate with the DASCON using the protocols and message formats as defined in the DAS/SWSI ICD. The SNIF, SDIF, and Isolator will work in tandem on one platform, with a second set of these subsystems running as a hot backup on a second platform.

There are two operational modes: normal and test. In normal mode NCCDS requests are sent to the operational NCCDS. The test mode provides access to the ANCC for performing EIF tests. This implies that a separate database instance is kept to test with ANCC. This database instance may have different data. Users will select test or normal operational use at login. No communications will be established with DASCON to support test mode. DAS requests will simply be stored in the test instance of the database to support DAS user training.

The system will also include a World Wide Web (WWW) server to view TDRSS Unscheduled Time (TUT). Currenty, the TUT information is only available to users on the Closed IONet. TUT information will be mirrored on the Open SWSI Server's web server where users can view this information.

2.5 Development Approach

2.5.1 Standards/Methodology

Since SWSI is an extension of the Jswitch system, much of the design already exists and will be reused, with the following exceptions:

- New common classes will be defined.
- A new Isolator will be developed.
- The Application Server will be mostly reused with some modifications and enhancements as previously discussed. The existing Application Server design will be used with the new JAVA components following the design of similar existing JAVA components.
- The Client application will be reused with extensive modifications to customize the user interface for SWSI. Since these modifications will only effect the placement of functionality (e.g. user login) within the user interface or will be an extension of the user interface, most of the existing design can be reused. New panels, subpanels, windows, or beans (as previously defined) will each be a separate SWSI class. Subpanels may get placed into subpackages if this provides some benefit.

Java development will follow coding standards originally developed by the NASA/GSFC Data Systems Technology Division in July 1997. These standards may be found at <http://aaaproduct.gsfc.nasa.gov/styleGuides/Java/Java.html>.

2.5.2 Development Environment

SWSI will be developed at GSFC Building 12, Room N12, which is a keycarded facility. All development personnel must have an appropriate security clearance in order to work on this project.

Development will be done on a Sun workstation running the Solaris (UNIX) operating system. Testing of the Client applications will also use Windows 98 platforms.

An /export/home/swsi directory will be created, under which the following subdirectories will be placed: dev, test, cm, and ops. This is shown in Figure 2-3. The subdirectories will follow the Java package naming standard. For SWSI, this will be *gov.nasa.gsfc.swsi*, with subpackages for the common classes and each of the subsystems. The Jswitch code will also be kept under the dev subdirectory for reference and to facilitate reuse. There will also be an /export/home/cots directory under which will be Phaos, JDK, and Infobus. The location of Oracle will be probably be loaded under the /export/home/cots directory. Finally, the Gnu's Not UNIX (GNU) C Compiler (GCC) and some security packages (tcp wrappers and ip filters) will be subdirectories under /usr/local/bin.

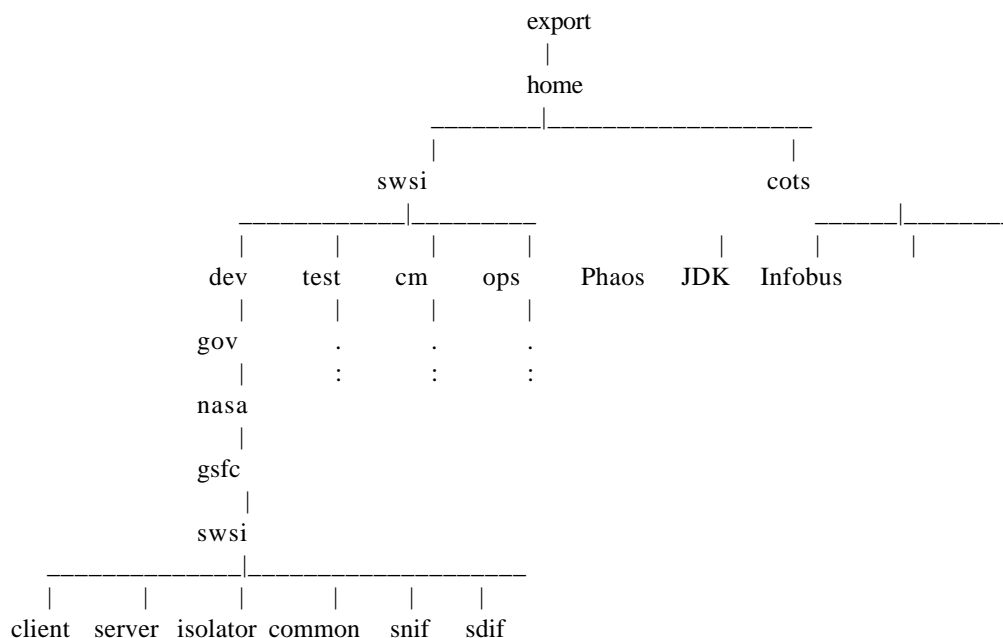


Figure 2-3 SWSI Development Directory Structure

Source code configuration management (CM) will use Concurrent Versions System (CVS), as used in another NCC project. CVS is based on Revision Control System (RCS). CM may reuse scripts from the Jswitch project.

2.5.3 Development Tools and Software

Graphics Designer Professional (GDPro) by Advanced Software Technologies, Inc. will be used as a design tool. GDPro will be used to create a selected subset of Unified Modeling Language (UML) diagrams including Use Cases, Sequence Diagrams, and Class Diagrams. These are given in the appendices.

The Integrated Development Environment (IDE) JBuilder Professional will be used. This tool provides the following capabilities:

- Source code editing
- GUI design and layout
- Rapid compilation and dependency checking
- Debugging

Entire subsystems may be built within JBuilder to allow use of the debugger. Since the debugger allows multiple debugging sessions, this can be used to test subsystem interaction.

JBuilder allows user defined JavaBeans to be placed on the component palette. These can include GUI beans or container classes.

Other support software will include:

- Operating System SunOS 5.7 (known as Solaris 2.7, also known as Solaris 7)
- Sun Professional Developer Suite (SunProWorkShop) - contains tools (i.e. compiler, debugger) for C application development
- Oracle Server (Release 8.1.6)
- Oracle Pro*C (Release 8.1.6.0.0)
- Java 2 Standard Edition Release 1.2.2 (free)
- HotSpot version 1.0.1 (free)
- InfoBus version 1.2 (free)
- Phaos SSLava Toolkit version 1.11
- Phaos J/CA Toolkit (for digital certificate generation) (For Build 1) to be replaced by NASA supplied Entrust Certificate
- Oracle supplied JDBC Thin Driver
- GNU tools: GNU C Compiler (GCC) version 2.95.2, GNU Debugger (GDB) version 4.18, Data Display Debugger (DDD) version 3.1.3
- TIBCO Extensibility TurboXML
- XML Parser (free)

Section 3. Client Design

3.1 Overview

The Client software is developed in Java and uses the Swing components of Java 2 to build a Graphical User Interface (GUI). JBuilder is used to build some of display panels. Most of the display panels have a window frame. At the upper right corner of the window frame there are three window control buttons: minimize, maximize, and exit. By clicking on these buttons, the user can iconify, uniconify, and close the display panel.

The Client subsystem has a Data Manager, which is responsible for establishing a secure connection with the SWSI Application Server. Each of the Client component panels will go through the Data Manager to send and receive data. The Client will obtain all the static data it needs from the Isolator through the SWSI Application Server using standard mnemonic requests. This static data will be used to present options to the operator. The displays that will be user customizable include the User Performance Data (UPD) displays.

The Client application will keep a time-tagged log of requests sent and messages received. This will include Ground Control Message Requests (GCMRs) and responses and acquisition failure messages, but not UPDs. Client events would also include when connections come up or down.

3.2 Use Cases

Since the SWSI is implemented using JAVA along with SQL and C, an object-oriented analysis and design (OOAD) methodology is the best choice for documenting all phases of software engineering activities. The Unified Modeling Language (UML) for use in OOAD provides all required tools to perform requirement analysis, system design, coding and testing.

Use-case modeling tool is used to describe what a new system should do or what an existing system already does. As shown in Figure 3-1 is a use-case diagram for the SWSI. In the object-oriented paradigm this diagram depicts a user view of the overall SWSI system-level functions and requirements. This view also exhibits interactions between SWSI and its user and administrator. All use cases are represented by ellipses. User and administrator are shown by actor symbols.

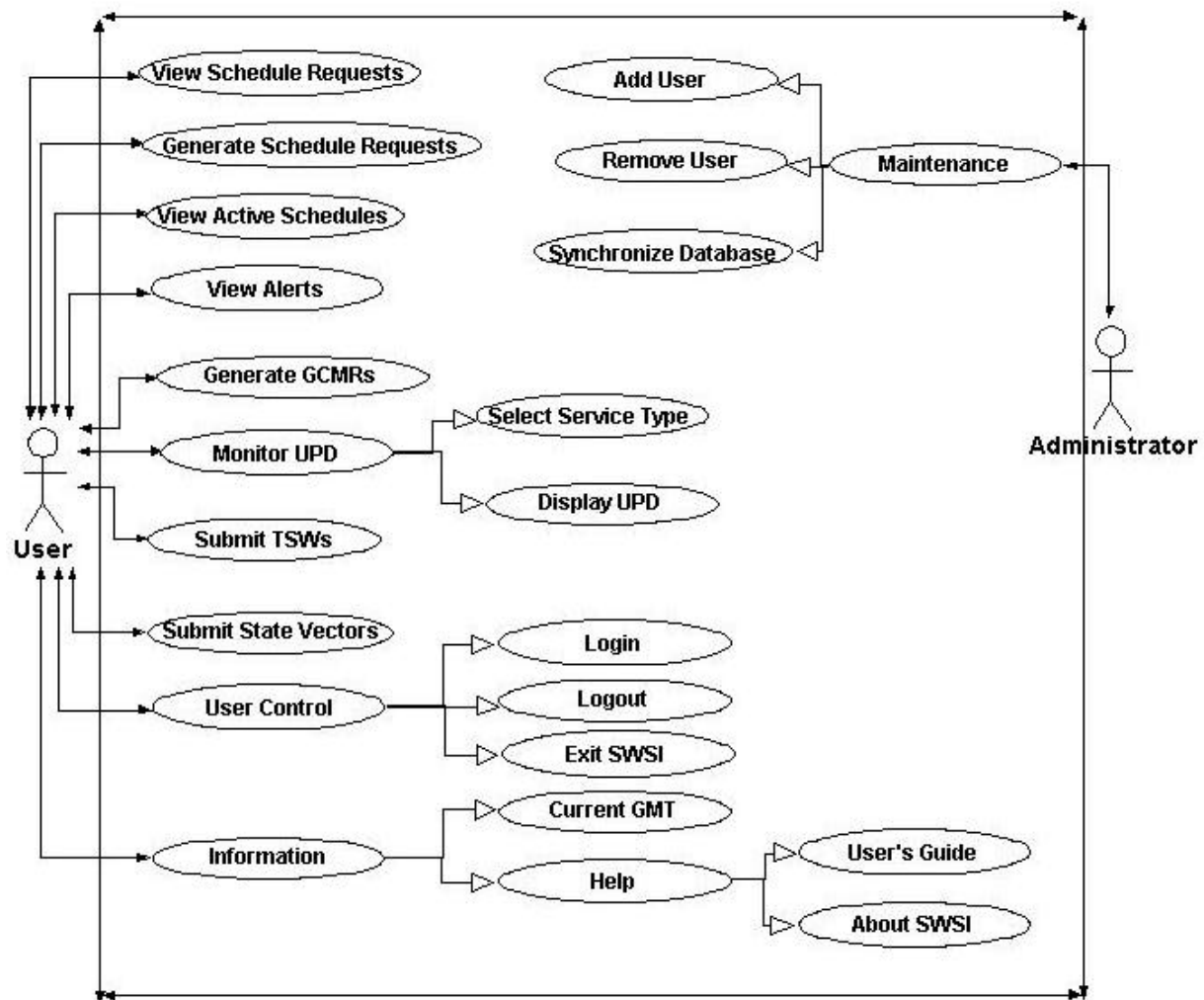


Figure 3-1 SWSI Use-Case Diagram

All the use cases for NCCDS support are described in Table 3-1.

Use Case	Description
View Schedule Requests	User can view all types of schedule requests including Schedule Add Requests (SARs), Replace Request, Alternate SARs (ASARs), Delete Requests and Waitlist Requests. Detailed functionality is described in Figure 3-2 and Table 3-2.
Generate Schedule Requests	User can generate any types of schedule requests. Detailed functionality is described in Figure 3-2 and Table 3-2.
View Active Schedules	User can view all active schedule events that have been accepted and approved by NCCDS. Detailed functionality is described in Figure 3-2 and Table 3-2.
View Alerts	User can view all levels of alerts issued by NCCDS or any other system sources.
Generate GCMRs	User can create and submit Ground Control Message Requests.
Monitor UPD	User can select Service Type; User can display UPD.
Submit TSW	User can submit TDRS Scheduling Windows.
Submit State Vectors	User can create and/or submit Improved InterRange Vectors (IIRV).
User Control	User can enter (login) SWSI Application Server by typing user ID, password, passphrase, and SWSI Application Server port number; User can disconnect (logout) SWSI Application Server; User can exit SWSI.
Information	User can see current GMT time; User can view SWSI help information including User's Guide; User can view information about SWSI.
SWSI System maintenance	Administrator can add user; Administrator can remove user; Administrator can synchronize SWSI database with NCCDS database.

Table 3-1. Use-Cases and Descriptions

Use cases (1), (2), and (3) are extended to show more details in Figure 3-2. Application Server, a SWSI subsystem, is an actor to provide retrieved data to be used by these cases.

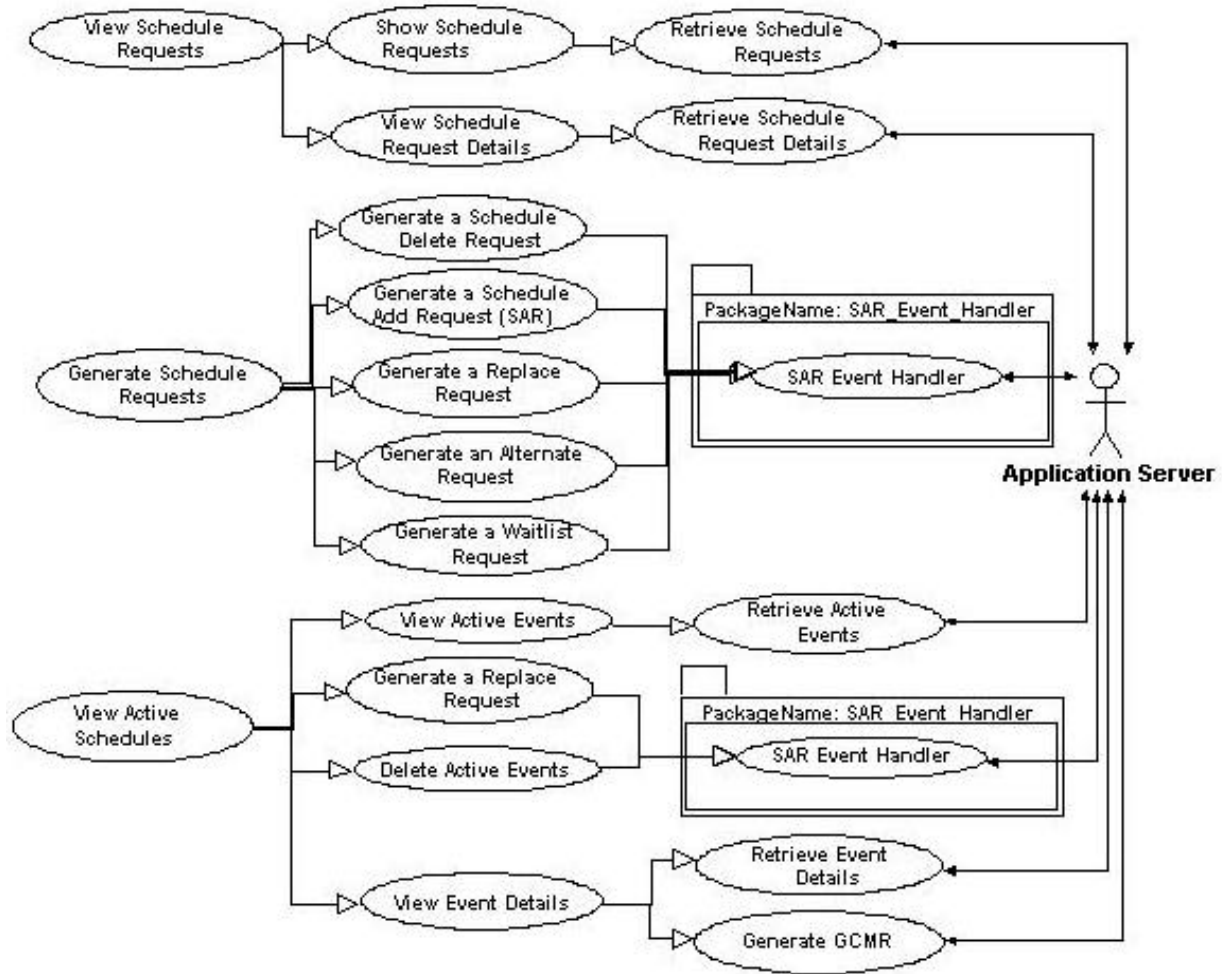


Figure 3-2 SWSI Extended Use-Case Diagrams

All extended use cases for NCCDS support are described in Table 3-2.

Use Case	Description
View Schedule Requests	This use case shows all schedule requests using data retrieved by SWSI Application Server; This use case allows user to select a schedule request and view its details that are retrieved by SWSI Application Server.
Generate Schedule Requests	This use case allow user to perform following tasks via an Event Handler Package that communicate with SWSI Application Server: Generate a Schedule Delete Request; Generate a Schedule Add Request (SAR); Generate a Replace Request; Generate an Alternate Request; Generate a Waitlist Request.
View Active Schedules	View all Active Schedule Events that are retrieved by SWSI Application Server; Select an Active Schedule Event and view its details that are retrieved by SWSI Application Server; Allow user via an Event Handler Package, which communicates with SWSI Application Server, to perform following tasks: Generate a Replace Request; Delete Active Schedule Events.

Table 3-2. Extended Use-Cases and Descriptions

The two use-case diagrams above document all SWSI functionality and requirements for subsequent development activities including design, coding and testing.

3.3 Client User Interface

3.3.1 Main Panel

Figure 3-3 shows the main panel layout. The main panel will include colored connection status indicators, showing the connection status of the Application Server, Isolator, SNIF and SDIF.

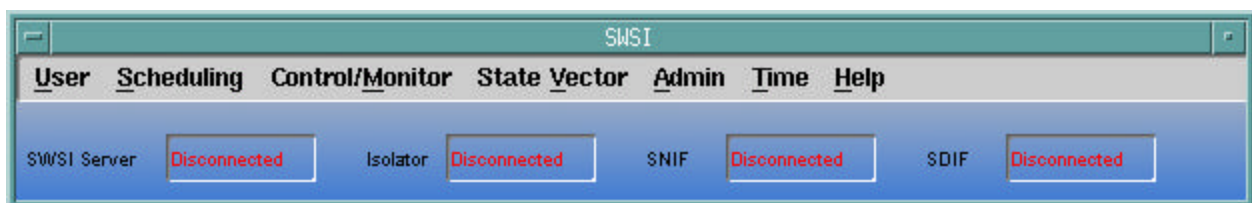


Figure 3-3 Main Control Panel

Figure 3-4 shows the menu options provided by the main panel. The NCC and DAS menu options are submenu titles giving access to NCC and DAS specific capabilities. The panel menu options will be disabled until a connection to the Application Server is made. The “Preferences” options allow the user to select between the standard Java look and feel settings. Only the supported look and feel options for that platform will be enabled.

SWSI						
User	Scheduling	Control/Monitor	State Vector	Admin	Time	Help
Log-in	NCC >	Alerts	Import	Edit SSCs	GMT Clock	User's Guide
Log-out	Create SAR	UPDs	Generate Stationary		Local Clock	About SWSI
Preferences	TDRS Scheduling Window					
Exit	DAS >					
	Resource Availability					
	Request					
	Create RAR					
	Playback Planning					
	Schedule Request Summary					
	Active Schedule Summary					

Figure 3-4 Main Control Panel Menu Options

The Scheduling, Control/Monitor, State Vector, and Admin menu items on the main control panel will be disabled until the user has logged in. After the user has logged in, these menu items will become enabled, allowing the user to select a panel for that mission.

The DAS specific options would only be available if the one or more of the SICs assigned to that user is flagged as being DAS enabled in the SWSI database (i.e., the menu options will be disabled for users with all non-DAS SICs). Likewise, a similar flag and restrictions would exist for NCC capabilities. This would allow a user to be strictly a NCC user, DAS user, or a combination of both. Finally, the Admin options would only be available if the user is flagged as a DAS mission administrator in the SWSI database.

All times entered or display will be in GMT as year, day of year, hours, minutes, and seconds in the form yyyydddhmmss.

3.3.2 Login Panel

Selecting Log-in results in the connection parameters panel being displayed, which is shown in Figure 3-5. This is the log-in screen for the SWSI Server. This panel contains text boxes for the following:

- Host - Shows the IP address of the Application Server (this option will be pre-set from the properties file and will be “grayed-out”, i.e., input will be disabled)
- Port – Shows the port on which the Application Server will be listening for Client connections (this option will be pre-set from the properties file and will be “grayed-out”)
- User Id – Enter the identifier assigned to the user to log on to the Application Server

- Password – Enter the password assigned to the user to log on to the Application Server. For security purposes, each password character will appear as an asterisk as it is entered.
- Passphrase – Enter the passphrase for the security certificate validation. This extra level of security validates that the user truly is authorized to access the SWSI Server

Additionally, the panel contains options for a normal or test (EIF) mode connection, and options to initiate a password and/or passphrase change.

Host
localhost
Port
3028
User ID
Password
PassPhrase
<input checked="" type="radio"/> Normal operational mode
<input type="radio"/> Test (EIF) mode
<input type="checkbox"/> Initiate Password Change?
<input type="checkbox"/> Initiate PassPhrase Change?
Last login:
Number failed login attempts:
<input type="button" value="Login"/> <input type="button" value="Logout"/> <input type="button" value="Done"/>

Figure 3-5 Connection Parameters Panel

The Last login displays the date of the last time a login to the SWSI Server was attempted and the number of failed login attempts since the last successful login.

At the bottom of the Connection Parameters Panel are buttons labeled *Login*, *Logout*, and *Done*. The functionality of each is described below.

Log-in

The Log-In button is used to establish a connection between the Client and Application Server. If the Client is not currently connected to the Application Server, the log-in button is active. After entering log-in information, clicking on the login button initiates a log-in attempt to the Application Server. Upon successful connection, the Application Server, Isolator, SNIF, and SDIF connection states displayed on the Main Control Panel turn from red *Disconnected* to green *Connected*. A Warning dialog box with the following message is also displayed:

This machine is connected to U.S. GOVERNMENT RESOURCES. If not authorized to access this system, disconnect now. YOU SHOULD HAVE NO EXPECTATION OF PRIVACY. By continuing, you consent to your keystrokes and data content being monitored.

Clicking *OK* closes the dialog box. Clicking *Cancel* returns back to the Connection Parameters Panel. If an Error dialog box is displayed due to the user entering invalid log-in information, clicking on *OK* closes the dialog box. After correcting erroneous log-in information, click on the log-in button again. Check that the entered host, port, user-id, and password information are correct in the event no connection can be made, an Error dialog box with the following message is displayed:

No connection could be made to host < host ip address> at port <port number>.

Log-out

Clicking on an active logout button disconnects the user from the Application Server. The log-out button is active if the label text is black (i.e. the Client is still actively connected to the Application Server). After the connection between the Client and Application Server has been terminated, the SWSI Server, Isolator, SNIF, and SDIF status boxes on the Main Control Panel turn from green *Connected* to red *Disconnected*.

Done

Clicking on the Done button removes the Connection Parameters Panel from the screen.

3.3.3 Schedule Request Panels

Schedule request panels permit the user to request SN resources and the NCCDS to schedule them.

The NCC Schedule Request Panels are made up of the Schedule Add Request (SAR) Panel, Schedule Delete Request (SDR) Panel, Alternate Schedule Add Request (ASAR) Panel, Replace Request (RR) Panel, and the Wait List Request (WLR) Panel. Two additional panels are used to support the SAR, ASAR, and RR panels. They are the Service Flexibility and Respecifiable Parameters Panels. All of these are discussed in sections 3.3.3.1 through 3.3.3.5.

DAS Schedule Request Panels consist of the Resource Allocation Request (RAR) Panel and the Resource Allocation Modification Request (RAMR). These are discussed in sections 3.3.3.6 through 3.3.3.7.

3.3.3.1 Schedule Add Request (SAR) Panel

Spacecraft events are scheduled by the NCCDS in response to a user's SAR. Each SAR designates a combination of support configurations in a particular time sequence for a specific duration. The design of the SAR Panel is shown in Figure 3-6.

The screenshot shows the 'Create SAR' panel with the following fields and controls:

- Message Class: SAR
- Request ID: 0000000
- Explanation: (empty text area)
- SUPDEN: A0018C5
- TORS: PSK
- ReferencedRequestID: None
- Priority: 1
- Buttons: Prototype Events, SSC, Add
- Table with 2 columns: Name, Type
- Table with 2 columns: Nominal Event Start Time, Freeze Interval
- Table with 2 columns: Plus Tolerance, Minus Tolerance
- Table with 2 columns: Wait List if unscheduled, Use TSWs to constrain scheduling
- Service Request table with 10 columns: Number, SSC, Service Type, Nominal Start, Nominal Duration, CSN, SRSN, (+)Tolerance, (-)Tolerance, Minimum Duration
- Buttons: Remove, Move Up, Move Down, Remove All, Modify Service..., Parameters...
- Buttons: Submit, Cancel

Name	Type
A01	MAF Normal
B01	MAR Normal
H01	SBAF Normal
H02	SBAF Normal
H03	SBAF Normal
H04	SBAF Normal
H05	SBAF Normal
None	SBAF Normal

Number	SSC	Service Type	Nominal Start	Nominal Duration	CSN	SRSN	(+)Tolerance	(-)Tolerance	Minimum Duration
001	A01	MAF Normal	00:00:00	00:01:00	000	000	00:00:00	00:00:00	00:01:00
002	B01	MAR Normal	00:00:00	00:01:00	000	000	00:00:00	00:00:00	00:01:00
003	T01	Tracking Normal	00:00:00	00:01:00	000	000	00:00:00	00:00:00	00:01:00

Figure 3-6 Schedule Add Request Panel

The SAR Panel will be used in viewing previously scheduled requests; “cloning” previously scheduled requests; or, generating new requests. Viewing a previously scheduled request involves selecting a request from the Schedule Requests Panel and pressing the “View...” button. Once this occurs, the Schedule Request Panel invokes the SAR Panel passing along as an argument the SAR object (see class diagram) associated with the selected request. The SAR is then displayed with all fields disabled so that no updates are made to the request. Cloning a previously scheduled request involves selecting a request from the Schedule Requests Panel and pressing the “Clone” button. Doing this invokes the SAR Panel passing along as arguments the SAR object associated with the selected request and a flag indicating that the intent is to “clone” an existing SAR object. New SARs may then be created using the existing SAR’s values as defaults. Generating a completely new request involves either selecting the “SAR” menu item from the Main Control Panel. The SAR Panel is invoked without any arguments allowing a user to generate a SAR.

To generate a SAR the user would first select a SUPIDEN from the drop-down list shown on the panel. This list of SUPIDENs is part of the static data sent to the Client application at login. Specifically, a vector of SUPIDENs is retrieved from the user's SetupObject using the DataManager's getSupidenList method. The list of available SUPIDENs is based upon the user, meaning only the SUPIDENs available to that user will be shown in the list.

The lists of service specification codes (SSCs) and prototype events are all dependent on the SIC, which can be obtained from the SUPIDEN selected. These lists (vectors) are obtained from the user's SetupObject using the DataManager's getSSCList and getPrototypeEventList methods. The list of TDRS IDs is independent of the SIC and system id (NCCDS/DAS). This list is obtained from the user's SetupObject using the DataManager's getTdrsIds method.

The Request ID value field is disabled with an initial value of "0000000" when generating a SAR. A SAR's Request ID is assigned by the Isolator.

The user may add a prototype event to the SAR by pressing the "Prototype Events" radio button and selecting a prototype event from the available list. Upon doing so, the "Service Request" label is renamed to "Prototype Event". Only one prototype event may be added to the list of services. Note that for prototype events, the services cannot be modified in any way.

HDS: In contrast to adding a prototype event, the user may add an SSC to the SAR by pressing the "SSC" radio button and selecting an SSC ID from the available list. Doing so populates the list of services with a service number, SSC ID, service type, and service flexibility parameters as described below for the selected SSC. Up to a total of 16 SSCs may be added to the list of services for a particular SAR. Services are added in an incremental order beginning with "1" on a first-selected, first-added basis. However, services may be reordered. To reorder a service, select the service to be moved from the Service Request list and press either the "Move Up" or "Move Down" button. The selected service's location in the list changes depending on which button was selected and all affected services are renumbered according to the selected service's new location.

Service respecifiable parameters are used in the SAR to change the initial values of certain data items in one or more SN services. The services may be modified by selecting an SSC and pressing the "Parameters..." button. This invokes a panel (see Figure 3-7) where parameter values for the selected service may be respecified, thereby overriding the initial values of the data items in the service.

The screenshot shows a software window titled "KASAR" with a list of configuration parameters on the left and their corresponding controls on the right. The parameters and their controls are as follows:

- TSW Set ID: [Text Input]
- Data Rate, I Channel (DG1/DG2), Normal User: [Text Input] bps
- Data Rate, Q Channel (DG1/DG2), Normal User: [Text Input] bps
- Transmit Frequency, Normal User: [Text Input] Hz
- Polarization, Normal User: ☒ LCP, ☐ RCP
- Maximum EIRP, Normal User: [Text Input] dBW
- Minimum EIRP, Normal User: [Text Input] dBW
- Autotrack Enable/Disable: ☒ Enable, ☐ Disable
- I/Q Channel Power Ratio (N:M), Normal User: [Text Input] dB
- Data Format, I Channel (DG1/DG2), Normal User: ☒ NRZ-L, ☐ NRZ-M, ☐ NRZ-S, ☐ Biphas-L, ☐ Biphas-M, ☐ Biphas-S
- Data Format, Q Channel (DG1/DG2), Normal User: ☒ NRZ-L, ☐ NRZ-M, ☐ NRZ-S, ☐ Biphas-L, ☐ Biphas-M, ☐ Biphas-S
- Data Bit Jitter, I Channel (DG1/DG2), Normal User: ☒ None, ☐ 0.01%, ☐ 0.1%
- [Unlabeled Section]: ☒ None

At the bottom of the window are "Cancel" and "Submit" buttons.

Figure 3-7 Example of respecifiable panel alone for KaSAR service

Service flexibility parameters may be modified by selecting an SSC and pressing the “Modify Service...” button. This invokes a panel (see Figure 3-8) where data items such as service start times relative to the SAR’s requested start time, service durations, plus and minus tolerances on the relative service start times, and minimum service durations may be changed. Additionally, this panel permits the user to specify the start of a service relative to the start of another service rather than relative to the SAR start time.

SUPIDEN	A0338CS	Service Number	001
SSC	A01	Request ID	9000142
Nominal Start	00	00	00
Nominal Duration	00	01	00
Plus Tolerance	00	00	00
Minus Tolerance	00	00	00
<input type="checkbox"/> Minimum Duration	00	01	00
<input type="checkbox"/> Coupled Service Number (CSN)	0		
<input type="checkbox"/> Service Bounded By Service Number (SBSN)	0		

Update Cancel

Figure 3-8 Edit Service Flexibility Parameters Panel

Upon completion of entering data for the SAR the user would press the “Submit” button on the SAR panel. This packages the data into a SAR object as seen in the SAR class diagram (see Appendix A), and transmits the object to the Isolator via the Server using the DataManager’s sendObject method.

3.3.3.2 Schedule Delete Request (SDR) Panel

An SDR permits a user to request the NCCDS to delete a scheduled spacecraft event or a schedule request.

To generate an SDR for a scheduled spacecraft event the user would select the event from the Active Schedule panel (Figure 3-13) and press the “Delete” button. Upon confirming the deletion an SDR object would be transmitted using the sendObject method. The SDR object would contain the message class, request ID (unknown until the request is transmitted by the SNIF), reference ID of the event to be deleted, and SUPIDEN. To generate an SDR for a schedule request the user would select the request from the Schedule Requests panel (Figure 3-11) and press the “Delete” button. Similarly, upon confirming the deletion an SDR object would be transmitted using the sendObject method. The SDR object would contain the message class, request ID (unknown until the ID is assigned by the Isolator), reference ID of the schedule request to be deleted, and SUPIDEN.

A user may view an SDR from the Schedule Requests panel. To do so, select the request whose status field indicates “Deleted” and press the “View...” button. The request would appear in an SDR panel as shown in Figure 3-9.

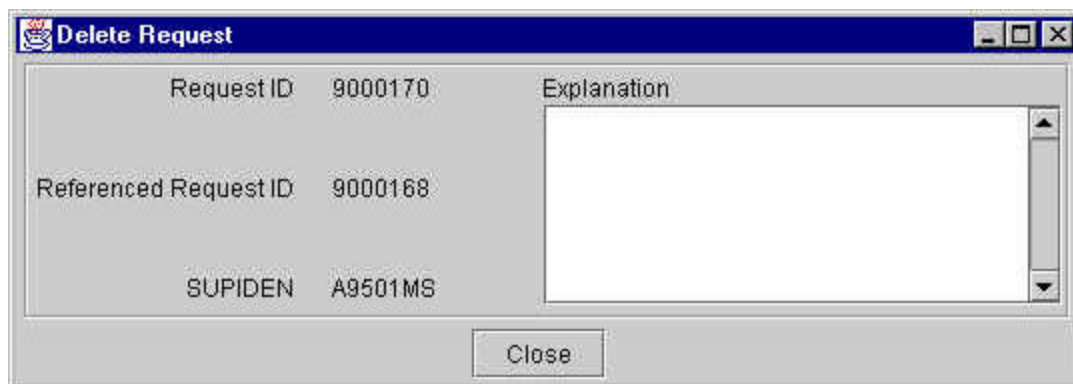


Figure 3-9 Delete Request Panel

3.3.3.3 Alternate Schedule Add Request (ASAR) Panel

The ASAR format is almost the same as the SAR format but allows for reference to a SAR, a Replace Request, or another ASAR queued for batch processing. Along that line the ASAR panel is nearly identical to the SAR panel except for the SUPIDEN, priority fields, and the “Wait List if unscheduled” flag, which will be “grayed-out”. ASARs inherit these fields’ values from the referenced requests and hence these fields are disabled on the ASAR panel. To create an ASAR a user would select the reference request from the Schedule Requests panel and press the “Generate Alternate...” button. A panel similar to the SAR panel (see Figure 3-6) would appear and the user would be allowed to modify the selected request’s information except SUPIDEN and priority. Upon completion of modifying the data the user would press the “Submit” button, packaging the data into an ASAR object (see ASAR class diagram, Appendix A) and transmitting the object to the Isolator using the sendObject method.

3.3.3.4 Replace Request (RR) Panel

The RR format is almost the same as the SAR format but allows for replacement of a scheduled event by another event or for replacement of a SAR, ASAR, or RR. Along that line the RR panel is nearly identical to the SAR panel except for the SUPIDEN and priority fields. RRs inherit these fields’ values from the referenced requests and hence these fields are disabled on the RR panel. To create an RR a user would select the reference request from the Schedule Requests panel or Active Schedule panel and press the “Generate Replace...” button. A panel similar to the SAR panel (see Figure 3-6) would appear and the user would be allowed to modify the selected request’s information except SUPIDEN and priority. Upon completion of modifying the data the user would press the “Submit” button,

packaging the data into an RR object (see RR class diagram, Appendix A) and transmitting the object to the Isolator using the sendObject method.

3.3.3.5 Wait List Request (WLR) Panel

The WLR refers to a declined request and requests that it be placed on the NCCDS Wait List.

To create a WLR a user would select the reference request from the Schedule Requests panel and press the “Generate Wait List...” button. This action would invoke a Wait List Request panel. This panel displays information about the request being generated and allows the user to modify the time that this request will expire and be removed from the wait list. Pressing the “Submit” button would package the information into a WLR object (see WLR class diagram, Appendix A) and transmit the object to the Isolator using the sendObject method. The design of the WLR panel is shown in Figure 3-10.

Figure 3-10 Wait List Request Panel

A user may also view an existing WLR by selecting the request on the Schedule Request Panel and pressing the “View” button. This would invoke a WLR Panel similar to Figure 3-10 except for the following changes. The Expiration Time field would be disabled not allowing user input; the Request ID value field would list the actual ID from the Schedule Request Panel; the “Submit” and “Cancel” buttons would be replaced with a “Close” button; and, the title would read “View Wait List Request”. The following figure lists the constructors used to invoke the WLR Panel.

WaitListReqFrame(SAR sarObject, boolean toView)	//if toView=true then view
WaitListReqFrame(RR rrObject, boolean toView)	// else create WLR

3.3.3.6 Resource Allocation Request (RAR) Panel

An RAR allows a DAS customer to request DAS resources. The design of the RAR Panel is shown in Figure 3-11. The constructors used to invoke the RAR Panel is as follows:

```
ServiceAllocationFrame(RAR rarObject, boolean toClone)
    //if toClone=true then clone RAR else view RAR
ServiceAllocationFrame(Date startTime, Date stopTime, String tdrs, String sic)
    //create RAR from selected DAS Availability Panel line
ServiceAllocationFrame()
    //create RAR by selecting menu item from Main Panel
```

The RAR Panel will be used in viewing previously scheduled RARs; “cloning” previously scheduled RARs; or, generating new RARs. Viewing a previously scheduled request involves selecting a request from the Schedule Requests Panel and pressing the “View” button. Once this occurs, the Schedule Request Panel invokes the RAR Panel passing along as arguments the RAR object associated with the selected request and a boolean flag set to “false” indicating that no cloning is to occur. The RAR is then displayed with all fields disabled so that no updates are made to the request. Cloning a previously scheduled request involves selecting a request from the Schedule Requests Panel and pressing the “Clone” button. Doing this invokes the RAR Panel passing along as arguments the RAR object associated with the selected request and a boolean flag set to “true” indicating that the intent is to “clone” an existing RAR object. The RAR Panel is displayed with the “Request ID” and “ReferencedRequest ID” fields cleared and a new RAR may then be created using the existing RAR’s values as defaults. Generating a completely new request involves either selecting the “Create RAR” menu item from the Main Control Panel.

The Request ID value field is supplied with an initial value of “0000000” when generating a new RAR; the Request ID is assigned later by the Isolator. The ReferencedRequest ID value field appears disabled with a value of “None” since this field is not used in an RAR. The user may modify other existing field values or enter new ones. However, to enable the SSC field for entry, the user must first select a SIC value. The latest DAS SSC parameter values will be retrieved from the database each time the user selects a new SSC. The user may modify any of the parameters for an SSC before submitting the request by pressing the “Modify” button, which becomes enabled after an SSC is selected. Pressing the “Modify” button invokes a ServiceParmWindow similar to Figure 3-7. Pressing the “Submit” button will cause the client application to check that the SIC, SSC, start and stop times are set before forwarding the request for scheduling.

Figure 3-11 DAS Resource Allocation Request Panel

3.3.3.7 Resource Allocation Modification Request (RAMR) Panel

An RAMR allows a DAS customer to modify a previously submitted resource allocation request. The constructors used to invoke the RAMR Panel are shown in Figure 3-12. The design of the RAMR Panel is shown in Figure 3-13.

ServiceAllocationFrame(RAR rarObject)	//create RAMR
ServiceAllocationFrame(RAMR ramrObject)	//view RAMR

Figure 3-12 Resource Allocation Modification Request Panel Constructors

The RAMR Panel is nearly identical to the RAR panel except that it contains the ReferencedRequest ID value of the RAR to be modified. The RAMR Panel's Request ID value field, too, is supplied with an initial value of "0000000" when generating a new RAMR; the Request ID is assigned later by the Isolator. RAMRs inherit their SIC and TDRS field values from referenced requests and these fields are not modifiable on the RAMR Panel. RAMRs also inherit their start/stop times and SSCs from referenced requests but these field values are modifiable. To generate an RAMR a user would select the reference request from the Schedule Requests panel and press the "Generate Replace" button. An

RAMR panel would appear and the user would be allowed to modify the selected request's information except SIC, TDRS, and ReferencedRequest ID. The user may modify SSC parameter values for the referenced SIC before submitting the request by pressing the "Modify" button. Upon completion of modifying the data the user would press the "Submit" button, packaging the data into an RAMR object (see Figure A-12 DAS Requests Common Class Diagram) and transmitting the object to the Isolator via the sendObject method.

DAS Resource Allocation Modification Request

ReferencedRequestID 8765432 RequestID 0000000

Service Start Time 2001 054 09 00 01

Service Stop Time 2001 054 12 00 00

SIC 0338 TDRS TDE

SSC RE1 Modify

Submit Cancel

Figure 3-13 DAS Resource Allocation Modification Request Panel

3.3.3.8 DAS Playback Modification Request Panel

If a DAS Playback Request is selected on a summary panel (see Sections 3.3.4 and 3.3.5) and the Generate Replace Button is pressed, the panel shown in Figure 3-14 would appear. This will allow the user to change the requested time for the playback to start, the destination of the playback, or the transmission protocol of the playback.

DAS Playback Modification Request	
SIC	<input type="text"/>
Event ID	<input type="text"/>
Old Start Time	<input type="text"/>
New Start Time	<input type="text"/>
Destination IP Address	<input type="text"/>
Destination Port Number	<input type="text"/>
Desired Transmission Protocol	<input type="radio"/> TCP <input checked="" type="radio"/> UDP
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

Figure 3-14 DAS Playback Modification Request

3.3.4 View Schedule Requests Panel

From SWSI Main-Panels menu bar user can select and open Schedule Request panel. In this panel, as shown in Figure 3-15, user can refresh data in the table by clicking Reload button. To view or delete any schedule request, user can first select any row in the table and then click View or Delete button, respectively. There are several types of requests displayed on the panel. Some examples are Schedule Add Request (SAR), Replace Request (RR), Alternate SAR (ASAR), and Wait List Request (WLR). The generation of these requests can be initiated from this panel by any of those four generation button. Finally, there is a Close button to allow user to close this panel.

Start Time	SURDEN	TDRS	Msg Class	Request ID	Status	Ref. Req ID	Creation Time
2000/102 05:00:00	A9501MS		Delete Req	9000170	Completed	9000168	2000/102 01:49:26
2000/102 05:00:00	A9501MS	046	SAR	9000168	Deleted		2000/102 01:49:27
2000/102 06:00:00	A9501MS	046	SAR	9000172	Completed		2000/102 01:52:37
2000/102 06:00:01	A9501MS		Waitlist Req	9000176	Completed	9000174	2000/102 02:01:18
2000/102 06:00:01	A9501MS	046	SAR	9000174	Expired		2000/102 01:53:08
2000/105 00:00:00	A0338CS	TDS	SAR	9000812	Activated		2000/098 17:42:37
2000/105 00:00:01	A1446DF	TDS	SAR	2231200	Activated		2000/098 15:45:49
2000/105 00:00:00	A0338CS	TDS	SAR	9000814	Declined		2000/098 17:51:21
2000/105 00:15:01	A1446DF	TDS	SAR	2231201	Activated		2000/098 15:45:51

Figure 3-15 Schedule Requests Panel

DAS Requests will be identified by unique Message Class identifiers to allow someone viewing these summary panels to distinguish between NCC and DAS requests. Additionally, someone viewing these panels could sort the requests by pressing the Message Class column header on that panel. This would group the requests by Message Class and allow the user to more easily identify DAS specific requests. DAS specific message classes are as follows:

- DAS RAR – DAS Resource Allocation Request
- DAS RADR – DAS Resource Allocation Deletion Request
- DAS RAMR – DAS Resource Allocation Modification Request
- DAS PBKR – DAS Playback Request
- DAS PBKDR – DAS Playback Deletion Request
- DAS PBKMR – DAS Playback Modification Request

To construct the Schedule Request Panel, ViewScheduleRequestFrame class is to define all required graphical user interface (GUI) components, data, and methods. As shown in Figure 3-16, column names and data of a table are declared and used to display all schedule requests. All GUI components are instances of Java and Swing beans, i.e., JPanel, JScrollPane, JButton, and JTable. A table model class, ViewScheduleRequestTableModel, is extended from a JAVA Swing default table model to construct the table object to be used in the frame class.

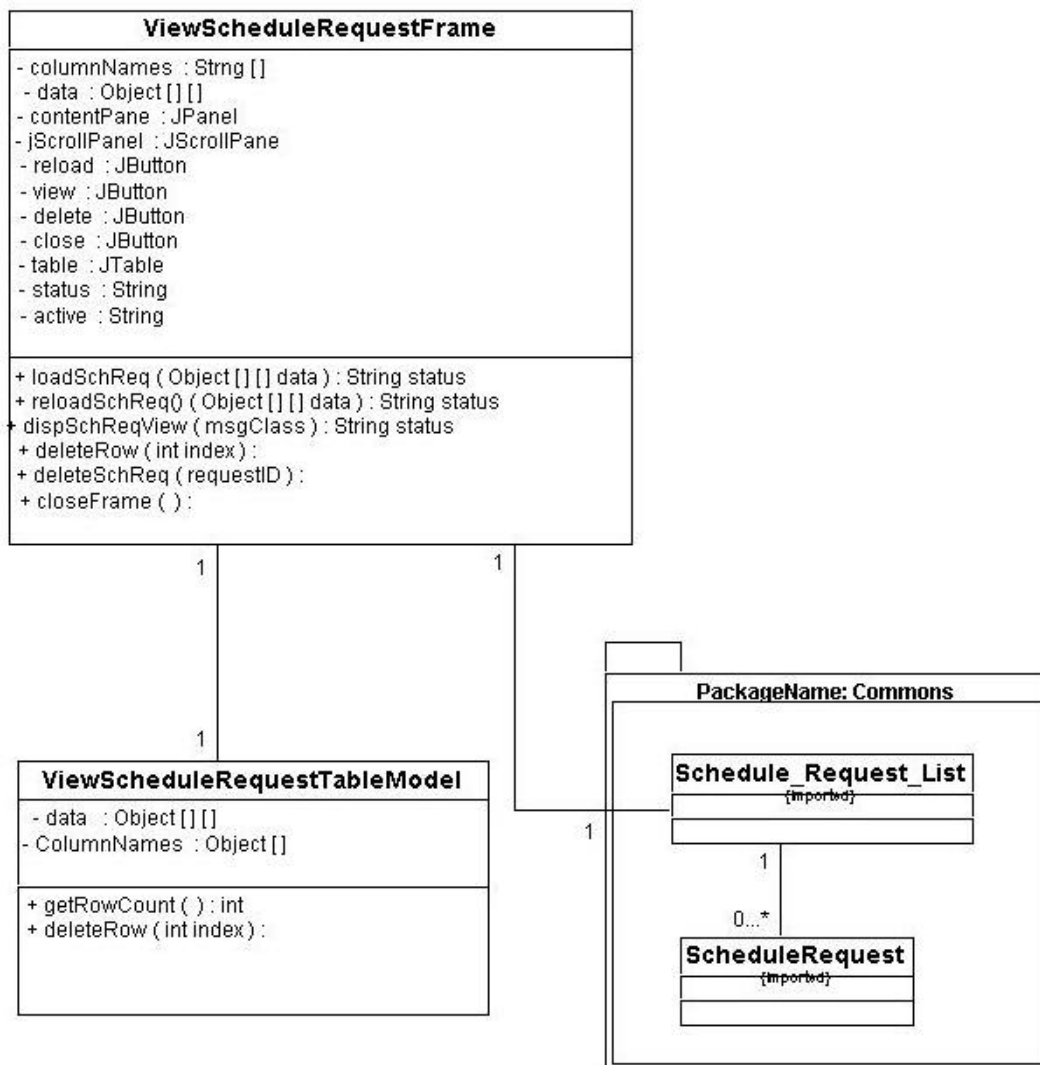
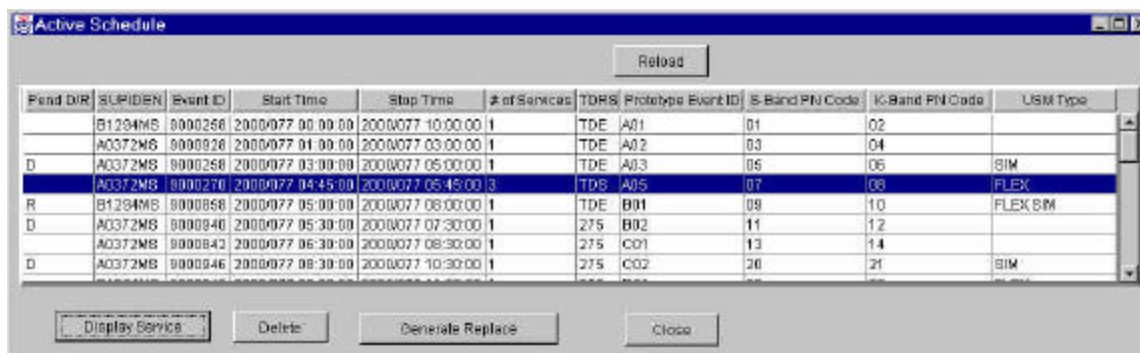


Figure 3-16 Class Diagrams for Schedule Requests

Each button click or a table selection invokes a correspondent action listener and a Java event handler to perform the required action. Function `loadSchReq` is a method to load data into the two-dimensional data array. To load data into the table, a request for `Schedule_Request_List` defined in the package, commons, is sent to Application Server. The Client's `DataManager` class will establish data storage, request data, and notify listener for data updated and/or data ready for access. All schedule request data are defined by class `ScheduleRequest`. Function `dispSchReqView` is called by function `reloadSchReq`. Function `deleteSchReq` is invoked after clicking the Delete button. Function `closeFrame` will close the frame after finishing schedule request activities. Note that user can only reopen this panel from the Main panel.

3.3.5 View Active Schedule Panel

From SWSI Main-Panels menu bar user can select and open Active Schedule panel. In this panel, as shown in Figure 3-17, user can refresh data in the table by clicking Reload button. To display service of any active schedule event, delete or replace any active schedule event, user can first select any row in the table and then click Display Service, Delete or Generate Replace button, respectively. Finally, there is a Close button to allow user to close this panel.



The screenshot shows a window titled "Active Schedule". At the top right is a "Reload" button. Below it is a table with the following columns: Pend DR, SUPIDEN, Event ID, Start Time, Stop Time, # of Services, TDRS, Prototype Event ID, S-Band PN Code, K-Band PN Code, and USM Type. The table contains several rows of data, with the row for Event ID 0000270 selected. At the bottom of the window are four buttons: "Display Service", "Delete", "Generate Replace", and "Close".

Pend DR	SUPIDEN	Event ID	Start Time	Stop Time	# of Services	TDRS	Prototype Event ID	S-Band PN Code	K-Band PN Code	USM Type
	B1204MS	0000266	2000077 00:00:00	2000077 10:00:00	1	TDE	A01	01	02	
	A0372MS	0000926	2000077 01:00:00	2000077 03:00:00	1	TDE	A02	03	04	
D	A0372MS	0000268	2000077 03:00:00	2000077 05:00:00	1	TDE	A03	05	06	SIM
	A0372MS	0000270	2000077 04:45:00	2000077 05:45:00	3	TDS	A05	07	08	FLEX
R	B1204MS	0000858	2000077 05:00:00	2000077 06:00:00	1	TDE	B01	09	10	FLEX SIM
D	A0372MS	0000840	2000077 05:30:00	2000077 07:30:00	1	275	B02	11	12	
	A0372MS	0000842	2000077 06:30:00	2000077 08:30:00	1	275	C01	13	14	
D	A0372MS	0000846	2000077 08:30:00	2000077 10:30:00	1	275	C02	20	21	SIM

Figure 3-17 Active Schedule Panel

DAS events will be given a USM Type of 'DAS RAR' or 'DAS PBK'. Like the Schedule Requests panel, someone viewing these panels could sort the events by pressing the USM Type column header on that panel. This would group the events by USM Type and allow the user to more easily identify DAS specific events. The blank USM Types are actually NCC fixed types and will be shown as 'FIXED' in the final implementation. Likewise, the 'SIM' types will be shown as 'FIXED SIM' in the final implementation.

Additionally, the logic for the summary panels would have to be modified to support the following functionality:

View Button (on Schedule Request Panel): The Schedule Request panel would have to be modified to call the correct class (DAS Request or SchAddReqFrame) depending on whether a DAS request or an NCC request was selected for viewing.

Generate Replace Button: Similarly for the Generate Replace button, if a DAS RAR request is selected, the DAS Resource Allocation Modification Request panel would be called. The panel would be shown with a title of 'DAS Resource Modification Request' and would include the original request's ID.

To construct the Active Schedules Panel, ActiveSchedulesFrame class is used to define all required graphical user interface (GUI) components, data, and methods. As shown in Figure 3-18, column names and data of a table are declared and used to display all the events in the active schedule. All GUI components are instances of Java and Swing beans, i.e., JPanel, JScrollPane, JButton, and JTable.

A table model class, `ActiveScheduleTableModel`, is extended from a JAVA Swing default table model to construct the table object to be used in the frame class.

Each button click or a table selection invokes a correspondent action listener and a Java event handler to perform the required action. The `loadUSM` function is a method to load data into the two-dimensional data array. To load data into the table, a request for `USM_List` defined in the package, `Commons`, is sent to the Application Server. The Client's `DataManager` class will establish data storage, request data, and notify listener for data updated and/or data ready for access. All active schedule event data are defined by class `USM`. Function `displayService` is called to open Service Display panel. Function `deleteUSM` is invoked after clicking the Delete button. Function `generateReplace` is called to open Create Replace Request panel. Function `closeFrame` is to close the frame after viewing active events. Note that user can only reopen this panel from the Main panel.

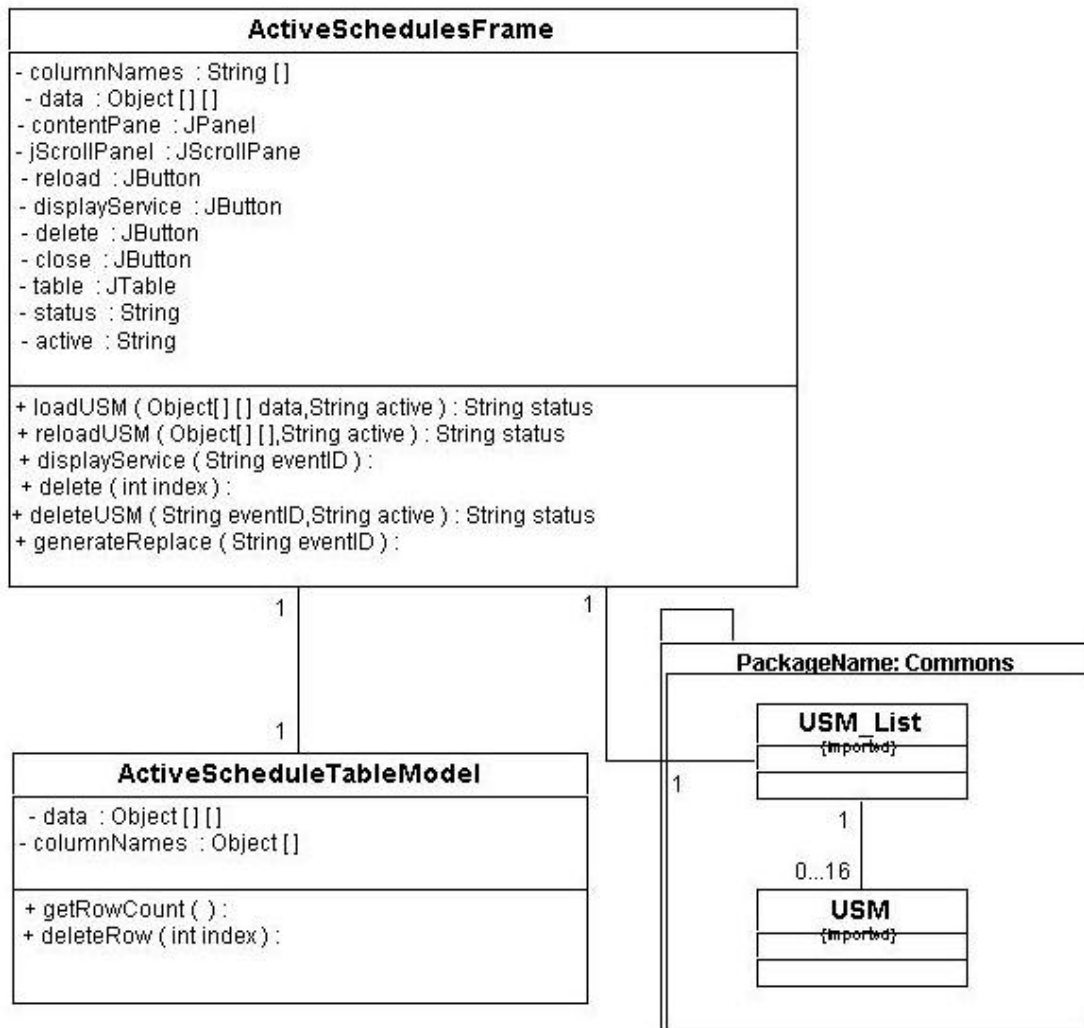


Figure 3-18 Class Diagrams of Active Schedule

3.3.5.1 View Service Display Panel

From Active Schedule Panel user can click Display Service button and open Service Display panel. In this panel, as shown in Figure 3-19, user can first select any row in the table and then click Parameters button. There is a Close button to allow user to close this panel.

Service Display

Supiden: A0372MS Event ID: 9000270

Start Time: 2000/077 04:45:00 TDRS: TDS

Stop Time: 2000/077 05:45:00 Prototype ID: A05

S-Band PN Code: 07

K-Band PN Code: 08

Service Type	SSC	Start Time	Stop Time	Link ID
SSAF Normal	HO2	2000/077 04:45:00	2000/077 05:45:00	2
SSAR Normal	HO2	2000/077 04:45:30	2000/077 05:45:00	2
Tracking Normal	HO2	2000/077 04:45:31	2000/077 05:45:00	

Parameters... Generate GCMR Close

Figure 3-19 Service Display Panel

A new button called 'View Handovers' will be added to the Service Display panel. This button would only be enabled when a DAS event is selected and the TDRS is marked as 'Any'. This button will bring up a subpanel displaying the planned TDRS handovers for that DAS service. This panel is a viewing only panel and is shown in Figure 3-19a.

DAS TDRS Handovers

SIC:

Service Start Time:

Service Stop Time:

Start Time	Stop Time	TDRS

Figure 3-19a DAS TDRS Handovers Panel

To construct the Service Display Panel, ServiceDisplayFrame class is to define all required graphical user interface (GUI) components, data, and methods. As shown in Figure 3-20, column names and data of a table are declared and used to display all the services. All GUI components are instances of Java and Swing beans, i.e., JPanel, JScrollPane, JButton, and JTable. A table model class, ServiceDataTableModel, is extended from a JAVA Swing default table model to construct the table object to be used in the frame class.

Each button click or a table selection invokes a correspondent action listener and a Java event handler to perform the required action. The loadUSM_Service function is a method to load data into the two-dimensional data array. To load data into the table, a request for USM_SSC_List defined in the package, Commons, is sent to Application Server. The Client's DataManager class will establish data storage, request data, and notify listener for data updated and/or data ready for access. All service data are defined by class ServiceRequest. Function ParametersDisp is called to open Parameters panel corresponding to the Service Type. Function closeFrame is to close the frame after viewing parameters activities. Note that user can only reopen this panel from the Active Schedules panel.

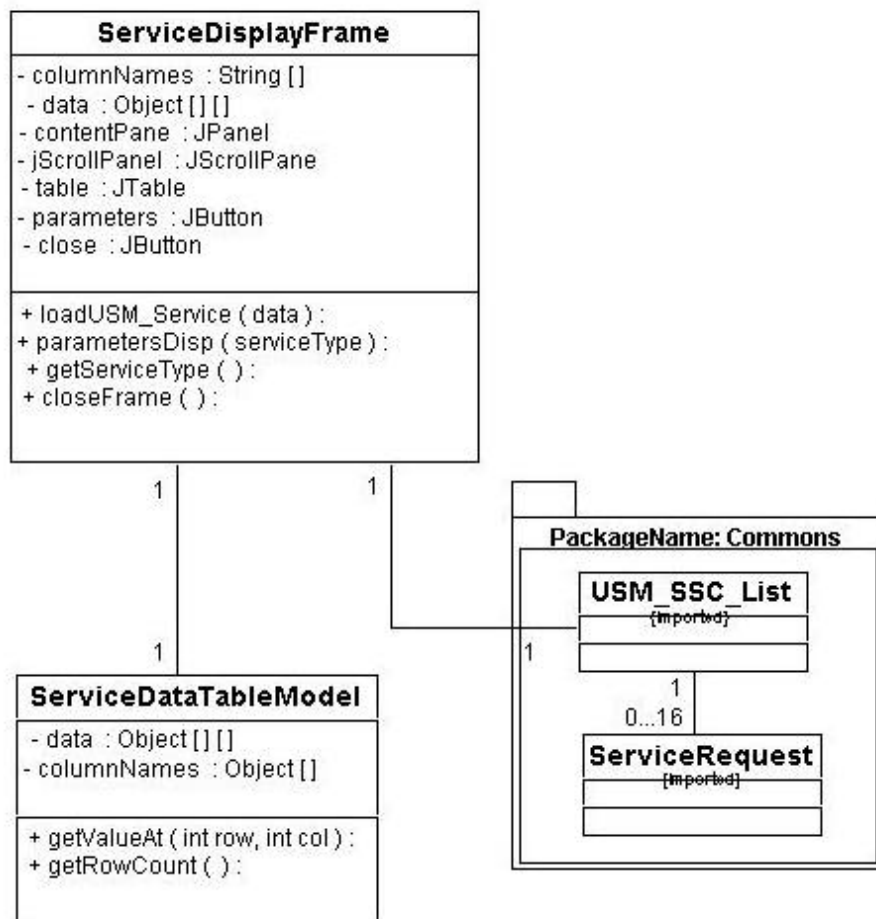


Figure 3-20 Class Diagrams of Service Display

3.3.6 Alert Panel

The alert panel will display NCCDS and DAS alerts and also alerts generated by SWSI subsystems. This will include:

- system status messages from the Application Server, Isolator, SNIF, or SDIF
- Schedule Result Messages (SRMs) with the result and explanation codes translated by the SNIF
- Acquisition Failure Notification messages
- The status and disposition of Ground Control Message Request (GCMR)

The alert panel will pop up whenever the first alert is received after the user connects to the application server. The user may also bring up this panel manually by selecting it on the SWSI main menu.

This panel will be implemented by modifying the current Jswitch event message panel. This alert messages panel, shown in Figure 3-21, will display alerts by color based on severity. A column will be added to show the source of the alert (Application Server, Isolator, SNIF, SDIF, or DAS). The panel includes a buffered, scrollable table that supports user selection of entries within the table. This allows the user to select a range of alerts to print or delete. The panel also provides optional alert logging to a flat file on the user's client computer.

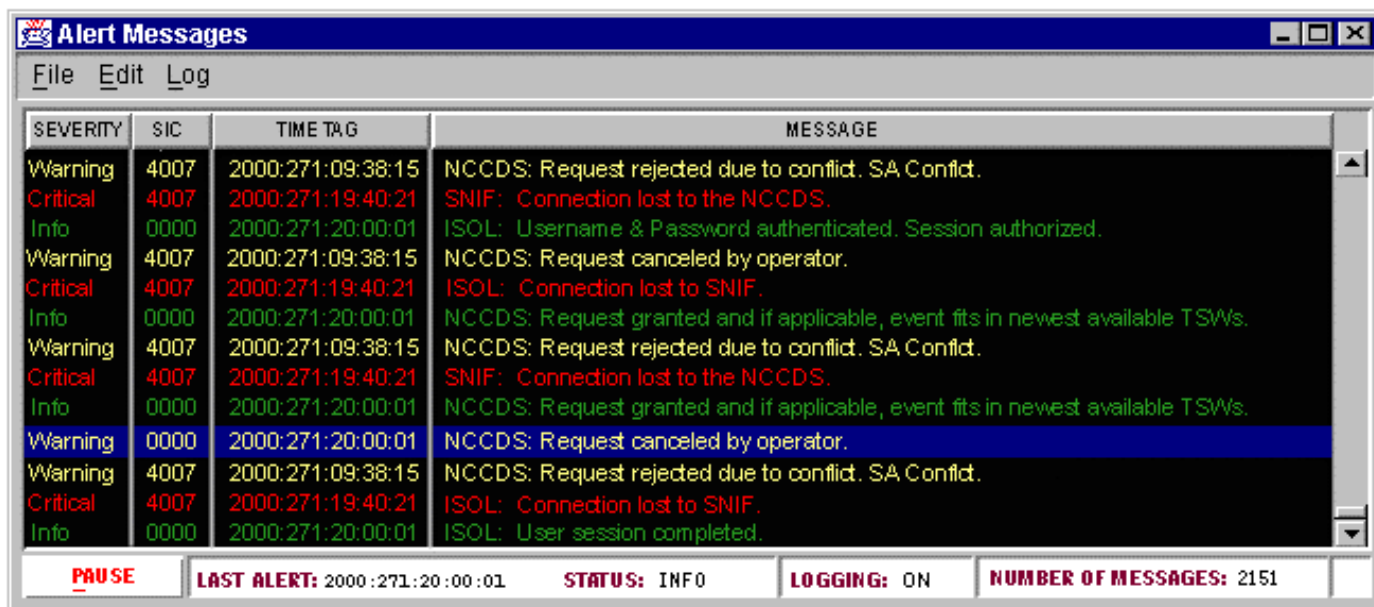


Figure 3-21 Alert Panel

This Jswitch event message panel will be modified to display and log SWSI alerts containing the following information:

- Severity (1 = information, 2 = warning, 3 = critical)

Severity	COLOR
1	Green
2	Yellow
3	Red

- Source (Application Server, Isolator, SNIF, SDIF, or DAS)
- SIC (Spacecraft Identification Code)
- TimeTag (YYYY:DDD:HH:MM:SS)
- Text of Alert contains the actual alert message

The panel will also be modified to change the panel title, remove the option to filter messages by message number or severity level, and to disable the menu option to “Open Event History Viewer”.

This latter option will be modified in some future release to support viewing of the alert history stored in the SWSI database.

Since alerts are routed by SICs and some identical alerts can be generated by multiple SICs and some clients can monitor multiple SICs, some clients may get multiple alerts.

3.3.7 GCMR Panel

3.3.7.1 GCMR Menu Panel for NCC Services

The Ground Control Message Request panel provides a means to submit ground control message requests (GCMRs).

The GCMR Menu Panel can be triggered by one of two methods:

1. Clicking on the GCMR button on the UPD Summary panel
2. Clicking on the GCMR button on the Service Display panel (a subpanel off the active schedule panel)

The GCMR Menu panel for NCC events will contain the following information: TDRS, SUPIDEN, Service Type, Link Number, and GCMR TYPE. Everything but the GCMR type will be pre-filled on the GCMR Menu panel.

Allowable GCMRs types will be hardcoded in the Client software. Valid values are as follows:

- Service Reconfiguration
- User Reacquisition Request
- Forward Link Sweep Request
- Forward Link EIRP Reconfiguration - Normal Power
- Forward Link EIRP Reconfiguration - High Power
- Expanded User Frequency Uncertainty Request
- Doppler Compensation Inhibit Request - none SSA

The pull down of GCMR types will include all allowable GCMR types for all services. It should be noted that some GCMR types are not valid for all Service Types. Invalid will be flagged by the NCCDS. Therefore, if the user selects an invalid GCMR Service Type/GCMR type combination, an alert generated by the NCC will be displayed on the Alert panel.

Clicking on the submit button at the bottom of the GCMR Menu panel for all GCMR types **except service reconfiguration** will send the following common objects to the Application Server.

GCMR Type	Common Object Sent
User Reacquisition Request	User_Reaction_Request
Forward Link Sweep Request	Forward_Link_Sweep_Request
Forward Link EIRP Reconfiguration – Normal Power	Forward_Link_EIRP_Reconfiguration (power mode set to Normal)
Forward Link EIRP Reconfiguration - High Power	Forward_Link_EIRP_Reconfiguration (power mode set to High)
Expanded User Frequency Uncertainty Request	Expanded_User_Frequency_Uncertainty_Request
Doppler Compensation Inhibit Request – none SSA	Doppler_Compensation_Inhibit_Request (compensationInhibitCode = none SSA)

Table 3-3. GCMR Type to Common Object Map

Clicking on the submit button at the bottom of the GCMR Menu panel after selecting the service reconfiguration value for GCMR Type will display a Service Reconfiguration panel. Service Reconfiguration GCMRs are discussed in detail in the next section.

3.3.7.2 GCMR Menu Panel for DAS Services

DAS GCMRs would use a DAS unique GCM menu panel (different from NCC requests). For DAS GCMRs, the EventID would be set, and the GCM types would be restricted to ‘Service Reconfiguration’ and ‘User Reacquisition Request’. The DAS GCM Menu panel is shown in Figure 3-22.

Selecting a ‘Service Reconfiguration’ will cause a ServiceParmWindow to be generated allowing the user to update the SSCs for that service.

Figure 3-22 DAS GCM Menu Panel

3.3.7.3 Service Reconfiguration Panel

If Service Reconfiguration is chosen as the GCMR type on the GCMR Menu, the ServiceParmWindow class described in Section 3.3.10 will send a GCParms_(EIF or norm)_(TDRS ID)_(supiden)_(ServiceType)_(LinkNumber) request to the DataManager (discussed in the common object area) to register with the DataManager class. The DataManager class will return a GCMR “placeholder” object. The ServiceParmWindow will setup a listener method on the GCMR “placeholder” object and wait for GCMR data containing the current ground control service parameter values for an event and service defined by the TDRS ID, supiden, and service type. The ServiceParmWindow class will take the layout specifications (static information such as labels and their locations <row and column>) and dynamically create the Reconfiguration panel. Refer to Section 3.3.10 for further details.

All reconfiguration panels will be divided into the following three sub-panels: the header sub-panel, the fixed parameter sub-panel, and the reconfigurable sub-panel.

The following fixed parameters will be displayed at the top of the reconfiguration panel in the header sub-panel:

- SUPIDEN
- Service Type
- TDRS
- Service Start Time
- Service Stop Time

The fixed parameter sub-panel will be displayed in the upper portion of the reconfiguration panel just below the header sub-panel. The reconfigurable subpanel will be displayed on the lower portion of the reconfiguration panel under the fixed parameter sub-panel. This panel will contain reconfigurable or modifiable parameters. Upon display of the reconfiguration panel, the fixed parameters, as well as the reconfigurable parameters, will be pre-filled with current service values.

The data on the header and fixed parameter panels will be displayed but “grayed out” (i.e., input will be disabled). Data on the reconfigurable parameters sub-panel can be modified in two ways:

- Reconfigurable parameters with a limited set of valid values: modify the current value by selecting a new value from the associated pull-down menu. To clear out a parameter, the user should delete the current value by using the delete key. All blank parameter values will be interpreted as no change to this parameter value.
- Reconfigurable parameters with numerous values: modify the current value by typing the new value into the corresponding text field; the new value is typed into an input field.

Clicking on the submit button at the bottom of Reconfiguration panel triggers the Service Reconfiguration class to load the Service_Reconfiguration_Request common object with the modified parameter values to be sent to the Application Server.

All GCMR Alert messages will be displayed on the Alert panel; no Alert information will be displayed on the GCMR panels.

3.3.8 UPD

When the user selects the UPD option from the Main Control Panel “panels” pull down menu, a UPD_(EIF or norm)_(SIC) common object is sent from a UPD driver class to the DataManager class (discussed in the common object area) to register with the DataManager class. The DataManager class will return a UPD “placeholder” object. The UPD driver will setup a listener method on the UPD “placeholder” object and wait for UPD data. A UPD Summary Panel that contains an empty Summary Active Services Table will be generated; the Active Services table will hold the UPD Summary information to be displayed on the Summary Panel. The Active Services table will be loaded as the DataManager begins to receive UPD data. The UPD Summary Panel section provides further details.

3.3.8.1 UPD Summary Panel

The UPD Summary panel contains a dynamically sized Summary Active Services table; the table will grow and shrink as services become active and inactive. The Summary Active Services table will be loaded with summary data for all active services per TDRS and SUPIDEN or SIC; a service is active if the DataManager is receiving UPD data. Each row in the Summary Active Services table will include the following service information: TDRS, SUPIDEN, UPD type, antenna or link number or DAS event id, and service status. A separate column of Ground Control Message Request (GCMR) buttons will allow the user to generate a GCMR for an active service. Below is a list of UPD types that can appear on the Summary Panel:

- SS AF
- KS AF
- KaS AF
- SSAR DG1
- SSAR DG2
- SSAR DQM
- SMAR DG1
- SMAR DG2
- SMAR DQM
- KSAR DG1
- KSAR DG2
- KSAR DQM
- KaSAR DG2
- KaSAR DQM
- MAF
- SMAF
- MAR
- MAR DQM
- SimF
- SimR
- KaSARWB DG2
- KaSARWV DQM
- DASMAR

The GCMR and service status columns in the UPD Summary Active Services table will be displayed as buttons. Clicking on a GCMR button triggers the GCMR subsystem. (Refer to Section 3.3.7 for details.)

The service status button will be labeled with the maximum severity of UPD detail data as determined by limit checks performed on the data from the UPD detail panel (ex. good, warning, out of tolerance or UPD ended). The color of the service status button will correspond to the label of the service status button as described in Table 3-4 below.

Color of Service Bar	Label	Meaning
Green	“Good”	Service is active. No limit-checked parameter failed.
Yellow	“Warning”	Service is active. At least one limit-checked parameter is at the warning level. No limit-checked parameters are worse than the warning level.
Red	“Out of Tolerance”	Service is active. At least one limit checked parameter is out of tolerance.
Gray	“UPD’s ended”	Service may not be active. Timeout expired without updates for this service.

Table 3-4. Service Status Button Coloring

Clicking on a service status button in the UPD Summary panel makes the detail panel for a TDRS/SUPIDEN or SIC/UPD type visible. If the detail panel is already visible, it is brought to the foreground.

DAS Services would be shown along with the NCC services in the UPD summary panel. The TDRS ID shown for a DAS service would be the current TDRS ID for that service, the SUPIDEN would be replaced by the SIC, and the link number replaced by the EventID. This UPD summary panel is shown in Figure 3-23.

User Performance Data Summary Panel						
Time	SUPIDEN or SIC	STATUS	Service	TDRS	Link or EventID	Submit GCMR
yyyydddhmmss	B1295MS	Good	MAF	170		
yyyydddhmmss	B1295MS	Out of Tolerance	SSAF	170	1	
yyyydddhmmss	1295	Good	DAS	046	9000928	

Figure 3-23 UPD Summary Panel

3.3.8.2 UPD Detail Panels

UPD detail panels displayed will either be standard UPD detail panels or customized UPD detail panels. Standard UPD detail panels will mimic the Operations Data Message (ODM) displays of the Communication and Control Segment (CCS) of the Network Control Center Data System (NCCDS). A generic translator program will take the specifications (static information such as labels and their location <row and column> found in the SetupObject and dynamically create a UPD detail panel; the SetupObject updDescList will be searched to locate the matching service and retrieve the UPD layout information. A sample UPD Detail Panel is shown in Figure 3-24.

DAS_MAR: UPD			
File Edit Execution			
TDRS	TDE	Service Start Time	2001\031 120833
SIC	2122	UPD Time:	2001\031 121031
Static Data			
Acquisition Mode:		Mode 1	
Carrier Uncertainty:		700 Hz	
Carrier Offset:		33 Hz	
	I Channel	Q Channel	
Data rate	75 bps	75 bps	
G2 Symbol Inversion	Inverted	Inverted	
Symbol Format	NRZ	NRZ	
Data Format	NRZ-L	NRZ-L	
PN Code	25	30	
Modulation	BPSK	BPSK	
Data Bit Jitter	None	0.01%	
Dynamic Data			
Chip Rate Estimate		150 bps	
Carrier Frequency Estimate		3300 Hz	
Eb/NO Estimate		>= 10-3	
Acquisition Frequency		3100 Hz	
	I Channel	Q Channel	
Symbol Rate Estimate	75 bps	75 bps	
Acquisition Time	2001\031 120833	2001\031 120833	
Lock Status	Lock	No Lock	
Loss of Lock Time	N/A	2001\031 121031	

Figure 3-24 UPD Detail Panel

Customized panels are standard panels that have been modified by the user and saved; customized panels are stored locally. Each service will have a UPD detail panel specified as default. The default UPD detail panel will be the panel displayed upon detail UPD panel selection. The user has the option to change the default detail UPD panel (Details are discussed below).

The UPD detail panels will have the following functionality:

1. Displays data only (ie, no user input/update)
2. Contains JavaBeans to provide reconfigurable components, limit checks of data and to set data background colors accordingly. The Range and Numeric beans will need options to set display color based on item values (refer to #5).

Bean	Type	Description
JBeanRange	Display	used to display an enumerated parameter type value
JBeanNumeric	Display	used for a numeric parameter type
JBeanText	Display	used for a text parameter type

Table 3-5. JavaBeans Needed for UPD Displays

3. Allows users to modify window layout. Assumes modified windows are only a subset of standard windows, not (for example) unions. (ex. delete mnemonics, delete components or labels, move mnemonics, move components or labels, properties, etc.) Clicking on the right mouse button will display a menu of user reconfiguration options. These modified panels detail will be stored locally in a file. There is no limit on the number of user reconfigured UPD detail panels.
4. Allows user specification of default UPD detail layouts, one layout per UPD type. The standard UPD detail panel will be the initial default UPD detail panel. Standard UPD detail panels are dynamically created using the UPD layout information in the SetupObject. Customized UPD detail panels, modified standard panels that have been stored locally, may also be set as the default panel. From a UPD detail panel menu, clicking on the set default option from the UPD detail menu bar will display a list of standard and user defined panels for the particular UPD type. Clicking on the name of a panel will set the chosen panel as the default UPD detail panel. The user can select (click on) either the standard panel which will be dynamically generated from the specifications given in the SetupObject or a locally stored user specified layout as the default UPD detail panel. If a user defined UPD detail panel is chosen as default, the UPD Type and window name will be written to a local file; no entry will be written to the file if the standard UPD detail panel is the default. When the user accesses the UPD subsystem, the default file will be read to determine which UPD detail panel is to be displayed. If a record is found in the default panel file, the panel specified will be displayed; if no entry is found in the default file for the specified UPD Type combination then the standard panel is dynamically created and displayed. Changing the default UPD detail panel will cause a readjustment of the UPD Summary panel service status button label and color, to reflect the maximum severity limit on the chosen UPD detail panel.

5. Enumerated values passed from the database will be translated into severity levels. Once the severity level has been established, the color associated with the severity level will be retrieved by performing a lookup in a table of severity levels mapped to colors. Icons may also be displayed adjacent to the data value to represent the severity level of the data item; the icon will aid operators who are color blind. The max severity level, of all data items, and its color will be displayed on the Summary Display service buttons (refer to Table 3-4 for details). The individual data severity levels will be shown by coloring the data field on the UPD detail panel as specified in Table 3-6.

Data Color	Meaning
Black	All Labels
Blue	Non Limit checked data item
Green	Limit-checked, data is in good range
Yellow	Limit-checked, data is in marginal range
Red	Limit-checked, data is out of tolerance or invalid

Table 3-6. Data Item Coloring

6. Displays UPD detail information in either 1 panel or 1 panel with 2 subwindows.

3.3.8.3 UPD Processing

As data is received, the UPD driver will determine the UPD type/SIC of the UPD data. A lookup in the Summary Active Services table will be performed to find a matching SIC/UPD type.

If a match of the SIC/UPD type is found in the Summary Active Services table the following will occur:

- An update method for the UPD type will be called; the update method will load the new UPD data values onto the existing UPD detail panel.
- A method will be called to update the max severity of the service status button on the Summary Panel.
- The UPD detail information will be parceled out to the appropriate JavaBeans within the corresponding UPD detail panel

If the UPD type is not found in the SIC Active Services table the following will occur:

- A look up in the default file by UPD type will be performed to determine which detail panel to create/open (a standard panel or a customized panel)
 - If the default panel is set to the standard UPD detail panel for that service, the UPD detail panel will be dynamically created. A look-up of the corresponding service in the setup object updDescList will be performed to obtain the layout of the panel. The UPD detail panel will be made invisible upon creation (the user will be able to display these panels by clicking on the service status button as described below). The reason behind creating the UPD Detail panels at

this point is to enable the max severity limit of the detailed data to be shown on the UPD Summary Panel. The detail data is then parceled out to the appropriate JavaBeans to fill in the UPD detail panel data values. Data value backgrounds will be set to an appropriate color. (Refer to Table 3-6 for details.)

- If the default panel is set to a customized panel, the panel is opened and the detail data is parceled out to the appropriate JavaBeans to update the data values in the UPD detail panel.
- Data value backgrounds will be set to an appropriate color. (Refer to Table 3-6 for details)
- The new service summary information is added to the UPD Summary Panel.
- The service status button on the Summary Panel will be set to reflect the max severity of the UPD detail data.

Timeouts of two types will be set on each UPD entry, row, in the Summary table. The first set of timers will be started when the UPD Summary panel is displayed. When UPD data is received by the DataManager class, the DataManager class will set the value in the UPD “placeholder” object, the UPD driver’s listener method will get called and the UPD driver will call a method in the appropriate panel to set the data. If no UPD updates have been received for a particular service within the specified time, the time value will be a configurable value that is stored and set in a property file, the timer will expire. Upon expiration of the timer, the label of the service button on the UPD Summary panel will be changed to “UPDs ended” and the service status button color will change to gray. At this time a second timer will be kicked off for that service. If, again, no UPD data is received within the specified time, the time value will be a configurable value that is stored and set in a property file, the second timer expires and the corresponding row in the UPD Summary panel table will be removed from the table. Also, all open corresponding UPD detail panels will be closed. If updated UPD data is received prior to the second timer expiration, the label of the service status button on the UPD Summary panel will be changed to reflect the maximum severity of all limit checks from the individual UPD display. The service status button will also be changed to the appropriate color as defined in Table 3-6.

3.3.9 TSW & State Vectors

TDRS Scheduling Window (TSW) messages notify the NCCDS of when the user spacecraft is within the line-of-sight of a TDRS. These line-of-sight periods may be further reduced by spacecraft attitude and antennae pointing limits, solar interference conditions, multipath interference conditions, or radio frequency interference (RFI) conditions such as the South Atlantic Anomaly (SAA). These predictions may be mission dependent and may require spacecraft ephemeris, solar ephemeris, attitude solutions, and spacecraft specific antennae limitations. SWSI will not have any prediction capabilities. Instead, it will have the ability to take prediction output in the format specified in Table 7-12 of the NCCDS/MOC ICD and form TSW messages that will be sent to the NCCDS through SWSI. The TSW panel may simply be a file chooser allowing the user to select which file of TDRS scheduling windows to be sent to NCCDS. A separate dialog box will be displayed asking the user to select a SIC for the TSWs being sent. TSWs are not sent to DAS.

The State Vector panels will support:

- 1 conversions from Latitude/Longitude/Altitude to type 8 IIRVs
- 2 IIRV direct entry
- 3 import of a file of vectors

The State Vector panels will also ask the user to select a SIC for the state vectors. SVs will be sent to both the NCC and DAS if the user was flagged as an NCC and DAS user, or to either NCC or DAS if the user was flagged as a user for only one of these.

If the user selects “Generate Vectors” from the Main Control Panel, the panel shown in Figure 3-25 will appear. The user would enter state vectors manually in this panel. If the user selects “Import” under “State Vector” from the Main Control Panel, a file chooser panel appears from which the user can specify which file to send.

State Vector Generation	
Geodetic Reference System WGS-84 Re = 6378.137 (Km), IFC = 298.2572	
Epoch Time:	<input type="text"/>
SIC	<input type="text"/> ▼
Data Source	Real-time ▼
Message Class	<input type="radio"/> Nominal <input checked="" type="radio"/> Inflight Update
Input Type	<input type="radio"/> Convert from Latitude/Longitude/Altitude <input checked="" type="radio"/> Direct IIRV Entry
Altitude: (Km)	<input type="text"/>
Latitude: [-90,90]	<input type="text"/>
Longitude: [0,360]	<input type="text"/>
X Position: (Km)	<input type="text"/>
X Velocity: (Km/s)	<input type="text"/>
Y Position: (Km)	<input type="text"/>
Y Velocity: (Km/s)	<input type="text"/>
Z Position: (Km)	<input type="text"/>
Z Velocity: (Km/s)	<input type="text"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

Figure 3-25 State Vector Input Panel

3.3.10 ServiceParmWindow

Subwindows for reconfigurable parameters (for GCMRs) and respecifiables or schedulable parameters (for Schedule Requests) will use a common window class. This class is the ServiceParmWindow.

This window will have a “header” portion and a split pane. The header portion will consist of the supiden, SSC, and service type for respecifiable windows; and the TDRS ID, supiden, service type,

and start and stop times (if available) for reconfigurable windows. The top pane of the split pane will contain fixed parameters. Fixed parameters will be displayed but “grayed out” (i.e., input will be disabled). The bottom pane will contain respecifiable or reconfigurable parameters, as needed. Both panes will be scrollable and will lay out parameters vertically. Figure 3-7 shows an early prototype of a window with the respecifiable pane alone. The final pane will differ in that a column of current values will be shown between the descriptions and the entry fields.

A pane of reconfigurable parameters will be similar. All parameters on the ServiceParmWindow will be displayed one per line, with a scrollbar if needed. For display allowing input, each parameter will have three columns displayed; a label column, current value/default value column, and a user input column.

The title of the final ServiceParmWindow will be:

- “<service type> Schedulable Parameters”, for respecifiable parameters
- “<service type> Reconfigurable Parameters”, for reconfigurable parameters

where an example of <service type> is “KaSAR”.

A window containing respecifiable parameters will have “Save” and “Clear” buttons. Hitting the “Save” button will cause the ServiceParmWindow to pass a Properties object back to the original caller. This Properties object will contain the set of keyword/value pairs defining the user selections and settings. “Clear” will reset all the user’s entries.

A window containing reconfigurable parameters will have “Submit” and “Cancel” buttons. “Submit” will cause a Service_Reconfiguration_Request common object to be created and sent to the Application Server via the DataManager. “Cancel” will cause the window to close without a common object being sent.

This class will support the creation of a standard display layout using the service description information provided by the login setup. An import function will take information from the login setup that defines the content and create windows.

This class will provide a new Service window. Depending on which constructor is used to create the window, it may have a submit button that uses data from GUI components within it to set and send an output object. Destination of output object will also depend on which constructor was used.

The following class diagrams show the interfaces and class definitions. A description follows each:

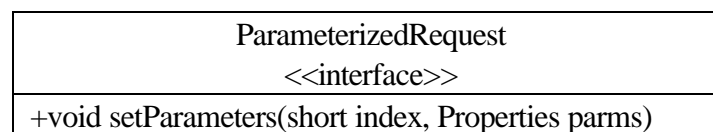


Figure 3-26 Class Diagram of ParameterizedRequest Interface

The ParameterizedRequest interface defines the method needed by the ServiceParmWindow class perform a callback to set the respecifiable parameter values. This allows the ServiceParmWindow object to return an object of the Properties class back to the caller. This Properties object will contain the keyword/value pairs defining the parameter settings made by the user. The ParameterizedRequest interface will be implemented by three types of panels: SAR, RR, ASAR, RAR, and RAMR.

ServiceParmWindow	
<<implements serializable, ActionListener, WindowListener, MouseListener>>	
-short schedFlag;	// 0=respecifiable input; 1=respecifiable display; 2=reconfigurable
-String myService;	
-String mySSC;	
-String myTdrsID;	
-String mySupiden;	
-Date myStart;	
-Date myStop;	
+ServiceParmWindow(ParameterizedRequest ref,	// for respecifiable input
short index,	
String service,	
String supiden,	
String SSC)	
+ServiceParmWindow(String service,	// for respecifiable display
String supiden,	
String SSC)	
+ServiceParmWindow(String service,	// for reconfigurable input
String tdrsID,	
String supiden.	
Date start,	
Date stop)	
-static ServiceParmWindow importService(String service)	
+void setParameters(Properties parms)	// to set existing parameter values

Figure 3-27 Class Diagram of ServiceParmWindow

The ServiceParmWindow class provides four possible constructors:

1. The first constructor which takes an argument of ParameterizedRequest, among others, generates a window of respecifiable parameters. This window will do a callback to the caller's (ParameterizedRequest object's) setParameters method to provide any parameter values that were set by the user when the user presses the "Save" button.
2. The second constructor, which takes arguments of service type, SIC, and SSC, generates a read only display of respecifiable parameters for the user to view. The calling object needs to first call this constructor, followed by a call to the setParameters method of this class to pass the current parameter values to the ServiceParmWindow for display. The "Save" and "Clear" buttons will be disabled.

3. The third constructor, which takes arguments of service type, TDRS ID, supiden, and start and stop times, will generate a display of reconfigurable parameters. The calling object needs to first call this constructor, followed by a call to the setParameters method of this class to set the current reconfigurable parameter values. The resulting ServiceParmWindow object will create and send a Service_Reconfiguration_Request object when the user presses the “Submit” button.
4. The fourth constructor, which takes an argument of Event ID, is used to support the DAS GCMR.

Each of these constructors work in a similar manner in that they will set the schedFlag to indicate what type of window is being created. They will then get the default window layout. If the default is a user defined layout, this window will be deserialized. If the default is the standard layout, the constructor will call the private, static method importService to obtain the service description parameters from the DataManager and create the window.

The ServiceParmWindow will also have the methods defined by the interfaces that it implements.

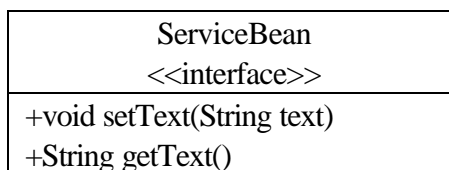


Figure 3-28 Class Diagram of ServiceBean Interface

The ServiceBean interface defines the bean methods needed for the ServiceParmWindow to get and set the text values of the GUI components within it. The types of beans needed include:

Bean	Type	Description
JBeanRange	Display	used to display an enumerated parameter type value
JBeanRadio	Input	set of radio buttons used for an enumerated parameter type
JBeanDropList	Input	drop down list used for an enumerated parameter type
JBeanNumeric	Either	used for a numeric parameter type
JBeanText	Either	used for a text parameter type

Table 3-7. JavaBeans Needed for ServiceParmWindow

Some of these beans may also be used for the UPD displays. Many of these beans will be simple extensions of the existing Java Swing beans to implement the ServiceBean interface. Beans implementing an enumerated type will have a property of JBeanStates[] which will map state names (i.e., text to display) to values. The display of Java Date values is TBD. Additionally, JLabel classes will be used to display labels within the window.

3.3.11 SSC Editing Panel

SWSI permits editing of existing DAS SSC parameters through the use of the SSC Editing Panel. To access this panel a user would select the “Edit SSCs” menu item from the Admin Menu of the SWSI Main Panel which invokes a panel similar to the one shown in Figure 29.

Enabling the SSC field for entry first requires the user to select a SIC value. The list of available SICs is retrieved using the DataManager’s getSICs method and consists of only those for which the user is flagged as having mission administrative privileges. After selecting a SIC value the user may select an SSC from the list of SSCs retrieved using the DataManager’s getSSCList method. The user may edit any of the parameters for an SSC by pressing the “Edit” button, which becomes enabled after an SSC is selected. Pressing the “Edit” button invokes a ServiceParmWindow containing the latest SSC parameter values retrieved from the database. Simultaneously, the client signals the Isolator to “lock” the SSC selected preventing other users from modifying the SSC’s default parameters. Once the user is done editing the parameter values and presses the “Submit” button on the ServiceParmWindow, the SSC parameter values are sent to the Isolator to be stored in the database, and the SSC is unlocked. The SSC is also unlocked if the user decides to abandon the editing session by pressing the “Cancel” button on the ServiceParmWindow or when user logs off.

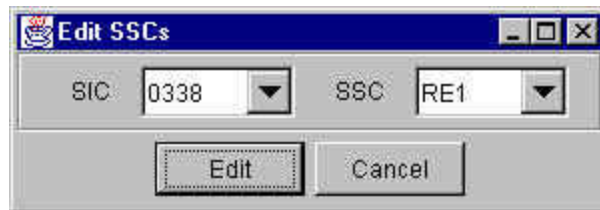
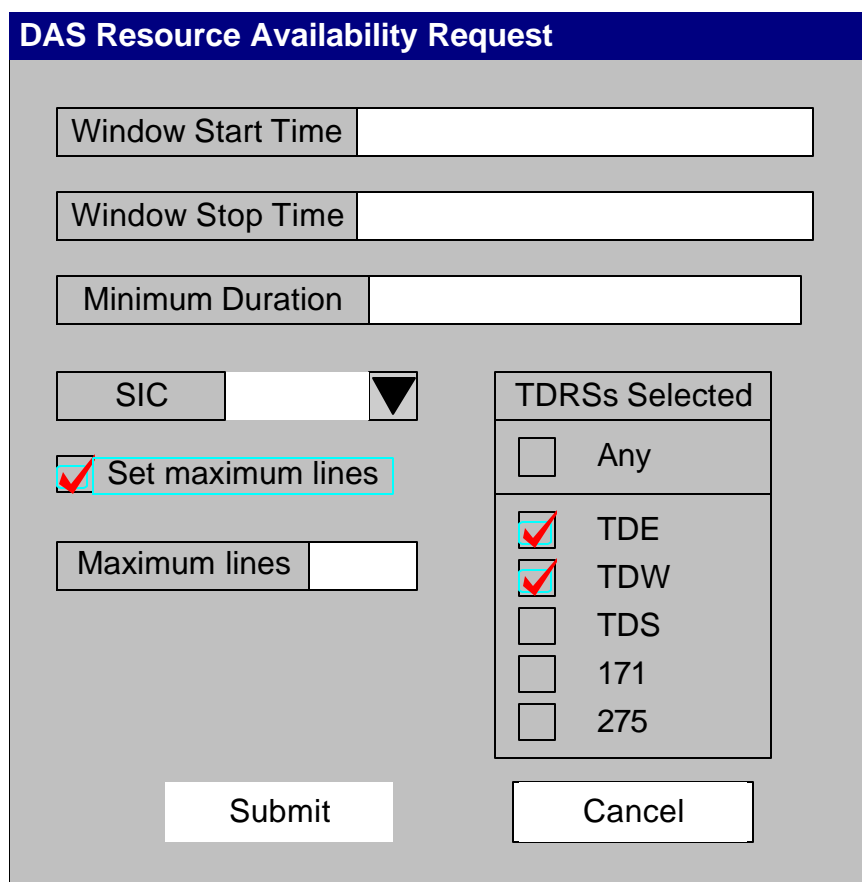


Figure 3-29 DAS SSC Editing Panel

3.3.12 DAS Resource Availability

Selecting the DAS ‘Resource Availability Request’ from the main control panel would result in a menu panel from which the user could choose to request a DAS resource availability report by specifying the time window within which the service is desired. This panel is shown in Figure 3-30. The ‘TDRSs Selected’ is a checkbox allowing multiple selections. The list of TDRSs would be created from the list provided in the SetupObject to the client. A user can select multiple TDRS or ‘Any’, which indicates no preference in selecting TDRSs for use. In this latter case, DAS would make the TDRS selection and indicate when if any TDRS handovers would occur.



The image shows a software panel titled "DAS Resource Availability Request". It contains several input fields and checkboxes. On the left, there are three text input fields labeled "Window Start Time", "Window Stop Time", and "Minimum Duration". Below these is a "SIC" label followed by a text input field and a downward-pointing arrow. Further down is a checked checkbox labeled "Set maximum lines" (highlighted with a red box), followed by a "Maximum lines" label and a text input field. On the right side, there is a section titled "TDRSs Selected" containing a list of options: "Any", "TDE", "TDW", "TDS", "171", and "275". The "Any", "TDE", and "TDW" options are checked with red checkmarks. At the bottom of the panel are two buttons: "Submit" and "Cancel".

Figure 3-30 DAS Resource Availability Request Panel

The DAS Availability panel, showing the resource availability report, is shown in Figure 3-31. This panel may contain additional columns indicating DAS resources available. This panel will contain a non-editable header showing the corresponding DAS Availability Request made by the user. The ‘Impact’ column shows, for dedicated users, what impact they might have on other missions by preempting this time slot. Impacts would be rated as being none, low, or high. A panel of this type will be created for

each availability request submitted. Multiple panels could be brought up to allow the user to compare availabilities during different periods.

DAS Availability

Window Start Time

Window Stop Time

Minimum Duration

TDRSs Selected

SIC

Impact	Start Time	Stop Time	Duration	TDRS	
					<div><div>▲</div><div></div><div>▼</div></div>

Create Request

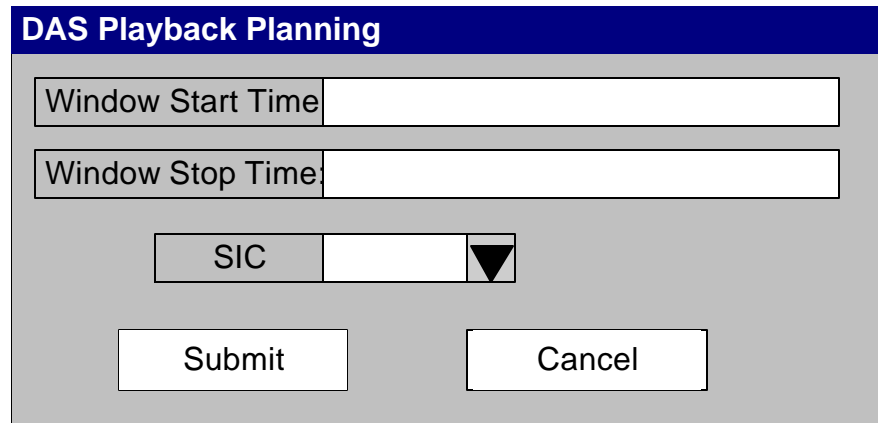
Cancel

Figure 3-31 DAS Availability Panel

The DAS Availability panel would allow the user to select a line and create a DAS Resource Allocation Request. If the ‘Create Request’ button is selected the DAS Resource Allocation request panel, discussed in section 3.3.3.6, is generated with the start time, stop time, desired TDRS, and SIC prefilled. Only one line can be selected per request.

3.3.13 DAS Playback Planning

Selecting the DAS 'Playback Planning' option from the main control panel would result in a menu panel in which the user could specify the time window within which data retrieval is desired. This panel is shown in Figure 3-32.



The image shows a software panel titled "DAS Playback Planning" with a blue header bar. Below the header, there are two text input fields: "Window Start Time" and "Window Stop Time". Below these fields is a label "SIC" followed by a small white input box and a downward-pointing triangle icon, indicating a dropdown menu. At the bottom of the panel are two buttons: "Submit" and "Cancel".

Figure 3-32 DAS Playback Planning Panel

The DAS Playback Availability report, shown in Figure 3-33, would allow the user to select and request an available playback. The user would be allowed to make multiple selections from the table as part of the same playback request. The DAS response to this request would be returned to the client in the form of an alert message.

DAS Playback Availability

Window Start Time:

Window Stop Time:

SIC

Start Time	Stop Time	Event ID	
			▲
			■
			▼

Desired Transmit Start Time

Destination IP Address

Destination Port Number

Desired Transmission Protocol

☐

☒

TCP

UDP

Create Request

Cancel

Figure 3-33 DAS Playback Availability Report

3.4 Data Manager

For the main panels the receive data (Active Schedule and Schedule Request), the main menu panel (or driver) will send the initial request to the DataManager, receive a ‘placeholder’ object from the DataManager, and set up a listener on this placeholder object. When data are received and the DataManager sets the value in the placeholder object, the driver’s listener method will get called and the driver will in turn call a method in the appropriate panel to set the data. Thus, each of these panels will need a **setValue(object)** method through which the common object can be passed once it is received.

To get updated data, the panels will call one of the following static methods in the DataManager:

- public static void getUpdatedUSMs();

- `public static void getUpdatedScheduleRequests();`

Subpanels making a request for detailed information will need to make a request to a DataManager static method, receive a 'placeholder' object, and set a listener on this object to receive an event signaling when the data are received. This is illustrated Figure 3-34.

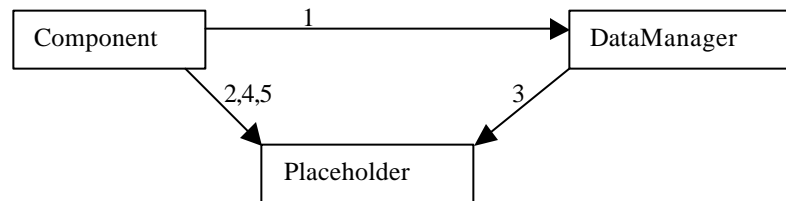


Figure 3-34 Data Request Procedure

- 1 The component will create a MnemonicRequest object. The class diagram for the MnemonicRequest class is given in Appendix A. This class provides data needed to define the different types of requests. The component only needs to fill in the subset of data needed for this particular request. The component will create a mnemonic name string using the naming convention given in Appendix A. This mnemonic name string also gets set in the MnemonicRequest object. If the component has previously requested data and created a MnemonicRequest object, the component can reuse this object to refresh ('reload') the data.
- 2 The component will call the static method: `DataManager.requestData(MnemonicRequest request);` which will return `DataValue` object (described later). Both the `DataManager` and the component now have a pointer to the placeholder.
- 3 The component will call: `addDataValueChangeListener(this);` which will add the component as a listener on that item.
- 4 Once the `DataManager` receives a value for that item, it will call `setObject(object);` setting the value in the placeholder.
- 5 The `DataValue` object will call the `dataValueChanged(DataValueChangedEvent)` method of all its listeners, alerting them that the placeholder now has a(n updated) value.
- 6 The component can now get the common object by calling the `getObject()` method on the `DataValue` (placeholder) object.

Figure 3-35 illustrates the event class and event listener interface.

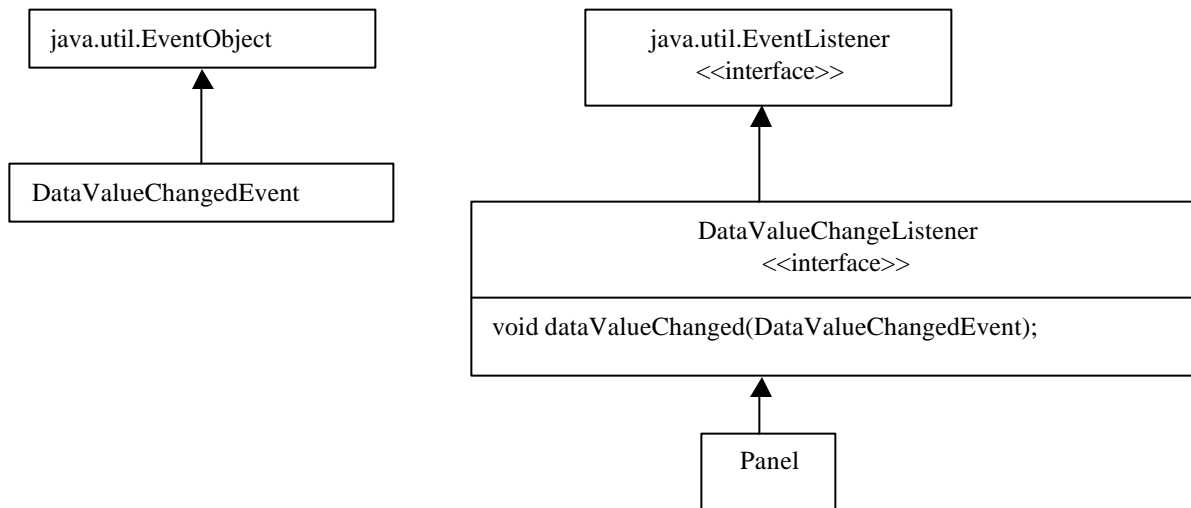


Figure 3-35 Event Class and Listener Interface

The DataValue class diagram given here lists its methods:

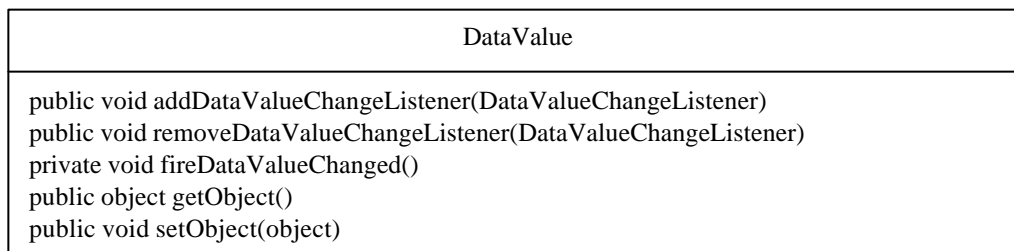


Figure 3-36 DataValue Class Diagram

The following class diagram show the methods available through the DataManager, including the methods used to retrieve the static data:

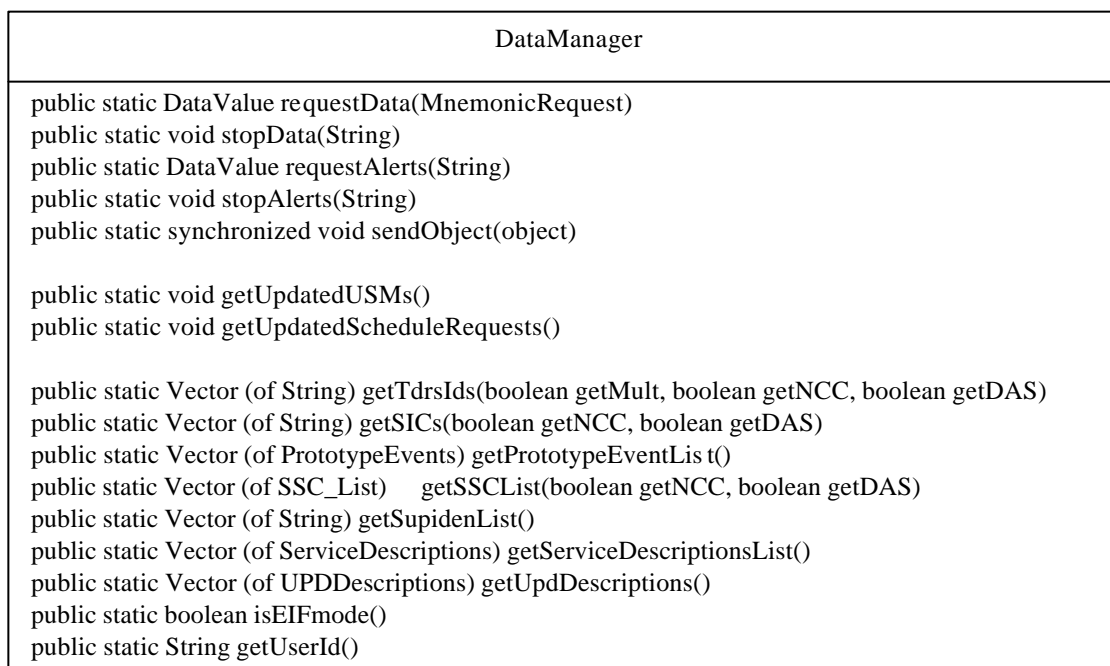


Figure 3-37 DataManager Class Diagram

3.5 Logging

All the alerts received will be logged in a file on the user's workstation. Other data e.g. , RCTD, and TTM data will be optionally logged. Other than this, no data, static or otherwise, will be stored on the Client PC due to security concerns.

Log files will use a standard naming convention. The following table defines this convention:

Name	Description
User_ID_#.log	Log file of alerts
(EIF or norm)_rctd_(date/time)_#.dat	return channel time delay data
(EIF or norm)_ttm_(date/time)_#.dat	time transfer message data

Table 3-8. Client Log Files Naming Convention

The Client subsystem's property file will contain a maximum size parameter for each of these files. A new log file will be opened when the maximum size is reached, with the _# number in the file name incremented by one.

Additionally, the Client can produce debug output to a separate debug output file. The file name and level of output are controlled by property values read in from the client's property file.

Section 4. Application Server Design

4.1 Overview

The SWSI Application Server is designed to perform three main functions: accept and provide authentication and security of Client connections, accept connections from the Isolator, and maintain the data flow to and from the SWSI Client. The Application Server will also log pertinent information. A standard Jswitch Application Server will be used which will be modified to add directives and messaging.

No validations or authentications (of, for example, SARs) are expected at the Application Server. This will all be done by the SNIF or Isolator.

The Application Server will have a backup on a separate platform. However, failure of the prime Application Server will necessitate that all the Client applications and the active Isolator reconnect to the backup Application Server and reinitialize their sessions. The use of a High Availability (HA) tool will allow the backup Application Servers to share the same IP address(es). This will allow the Clients and Isolator to connect to the same IP address regardless of which Application Server is primary.

The Application Server design is given in Figure 4-1.

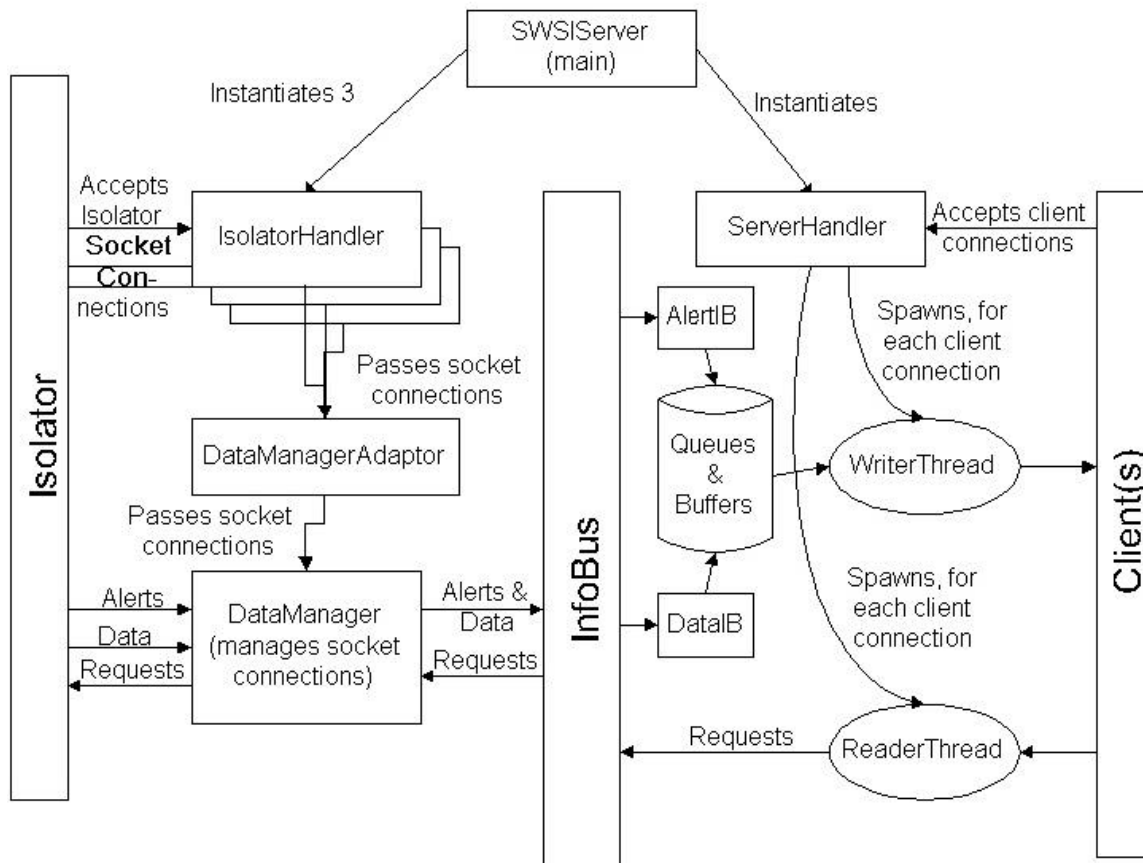


Figure 4-1 SWSI Server Design

The Application Server will receive all incoming Client requests to connect to SWSI. Client applications will establish an SSL connection with the Application Server and will exchange digital certificates with the Application Server for authentication. Client applications will then provide a user ID and password as part of a login object. This object will be forwarded to the Isolator for verification. The userID will be retained by the Application Server for later use. The Isolator will respond with a SetupObject if the user ID and password are verified, which the Application Server will forward back to the Client. Client connections will be accepted after connection to the Isolator machine is established.

Isolator connections will consist of 3 socket connections, each of which will be initiated by the Isolator. Three instances of the IsolatorHandler class will be created to accept these connections. Once these connections are made, the connections are handed off to the DataManager class via an instance of the DataManagerAdaptor class.

The DataManager class will be responsible for maintaining the connection to the Isolator machine. The DataManager will be responsible for routing requests to the Isolator. Client threads can send requests

to the DataManager, which will be queued for processing. The Data Manager will also maintain an instance count of each request and alert service. This is coordinated with the InfoBus.

The Application Server will maintain the data flow to the Client using the InfoBus. The InfoBus will keep a list of the requests associated with each Client thread. The InfoBus will use a notification mechanism to inform Client threads when requested data are available. The InfoBus will also distribute the alert messages to each Client thread associated with the alert's SIC.

4.2 Detailed Design

The SWSI Application Server will use an instance of the ServerHandler class to create a server socket that waits for incoming Client connections. It will then authenticate each Client connection using the SSL protocol and pass a username/password combination to the Isolator for authentication. Once a successful connection is made, the instance of the ServerHandler class will clone itself to create an instance that will further tend that specific Client connection. A separate clone is created for each Client connection.

Two new Client threads will be spawned by the clone. Thus, two threads will exist for each Client connection. A reader thread will read the socket connection and process requests. A writer thread will write request responses back to the Client. Both of these Client threads are within the scope of the ServerHandler clone. The clone tracks whether the Client has responded to previous data sent to the Client with an ack (DataRequest object) before sending additional data. The ack indicates the Client is ready to accept more data. This handshaking is designed to prevent the Application Server from filling its socket with data before the Client is able to accept it.

The ServerHandler class will be responsible for the connection to the Client. The SetupObject is sent by the Isolator through the Application Server to the Client upon successful login of the Client. The Client thread will handle all communication to and from the Client. Only Client specific information is maintained in these threads. The Client thread will communicate with the DataManager via the InfoBus to make requests to the Isolator and receive responses. The Client thread will not send a data update until the Client acknowledges the previous update. This is to prevent buffering old data, which could cause the Application Server to fail. The design will include a small FIFO queue of alerts, in the event the Client or network performance falls behind. Old alerts will be dropped from the queue if the queue reaches a maximum size. Alerts can be filtered by severity so important messages are not dropped.

When the Client requests alerts (which the Client will do automatically), the ServerHandler clone will create an instance of the AlertIB class. This class will receive alerts off the InfoBus and queue alerts for the SICs being monitored by that Client. Likewise, if the Client makes any other requests, the ServerHandler clone will create an instance of the DataIB class. This class will receive responses from the Isolator off the InfoBus and store the responses in local buffers. When either the AlertIB or DataIB classes have data, the writer thread will be resumed to write the queued or buffered data to the Client.

4.3 Data Interfaces

The Application Server will require the following input and output files.

Server Inputs

The Application Server will need digital certificates to authenticate itself to Clients and the Isolator, and will need the CA certificate to authenticate the Clients. The Server will also need property files that contain configuration parameters for the Application Server. The following table lists these:

Name	Description
other.prop	Provides non-SSL required properties such as port numbers
SSL.prop	Provides SSL specific properties (crypto-suite to use, etc.)
SWSI-ca-cert.der	Certificate authority certificate
SWSI-server-cert.der	Server Certificate
enc-SWSI-server-key.der	Encrypted Server key

Table 4-1. Application Server Input Files

Server Outputs

The Server will write to the following two logs:

Name	Description
bad_logins.log	Log of rejected login attempts
activity.log	Activity log

Table 4-2. Application Server Output Files

4.4 Logging

The Application Server will maintain two log files. All logins (successful or failed) shall be logged, and will include the IP address. The activities of a successful connection will be logged. The log will identify the time and Client connection along with the activity. The Application Server will be configurable to have some control over the granularity of logging other data, i.e., what types of requests will get logged. These logging options will be set as a properties in the Application Server's other.prop file. Other properties will include the names of the log files. A separate file will be maintained for the unsuccessful login attempts. In order to avoid the log files from growing forever, the Application Server create a new log file every xx days (xx will be controlled by a property in other.prop). This design will allow a system operator to delete the old log files. Additionally, the Application Server can produce debug output to a separate debug output file. The file name and level of output are controlled by properties in other.prop.

Section 5. Isolator Design

5.1 Overview

The Isolator Subsystem serves as the central communication node for all the other SWSI internal subsystems. These subsystems are the Application Server, SDIF, SNIF and the SWSI Database Server which are, in turn, dedicated communication nodes to the end external elements such as the NCC/DAS users, the DASCON, the NCC/ANCC and the SWSI databases. Figure 5-1 depicts the communication flow between the SWSI internal subsystems and external elements.

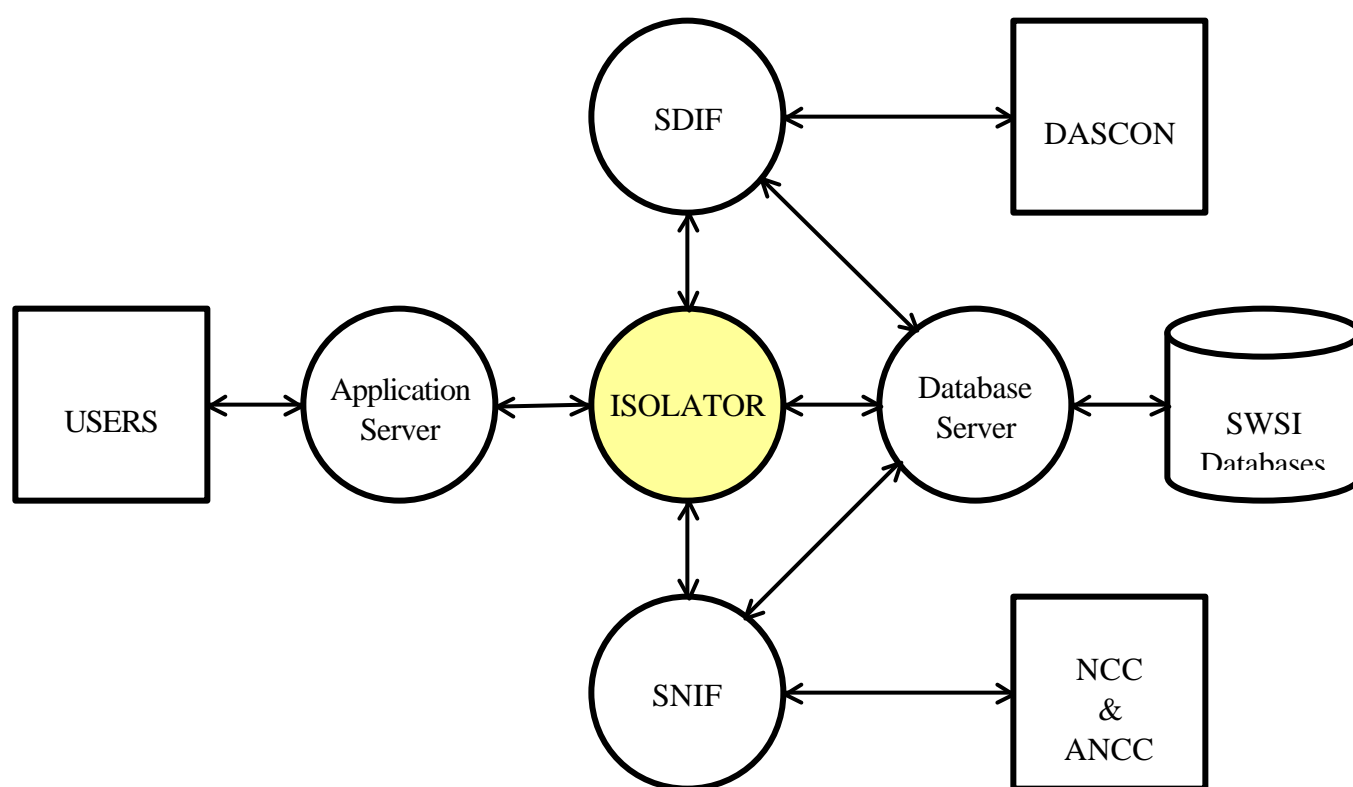


Figure 5-1 *Communication Flow of SWSI elements*

The primary functions of the Isolator are to receive user requests from the Application Server, to process the requests and to send responses back to the Server to be forwarded to the client. The Isolator also forwards messages to the NCC/ANCC or DASCON, passes Alert, TTM and UPD information to the Application Server, stores and retrieves data from the SWSI database and logs information about user and database activity. The Isolator subsystem resides on the same platform with

the SNIF subsystem, the SDIF subsystem and the SWSI database server (Figure 5-2). However, the Isolator design allows for the separation of any or all of those subsystems such that they can run on different platforms if needed.

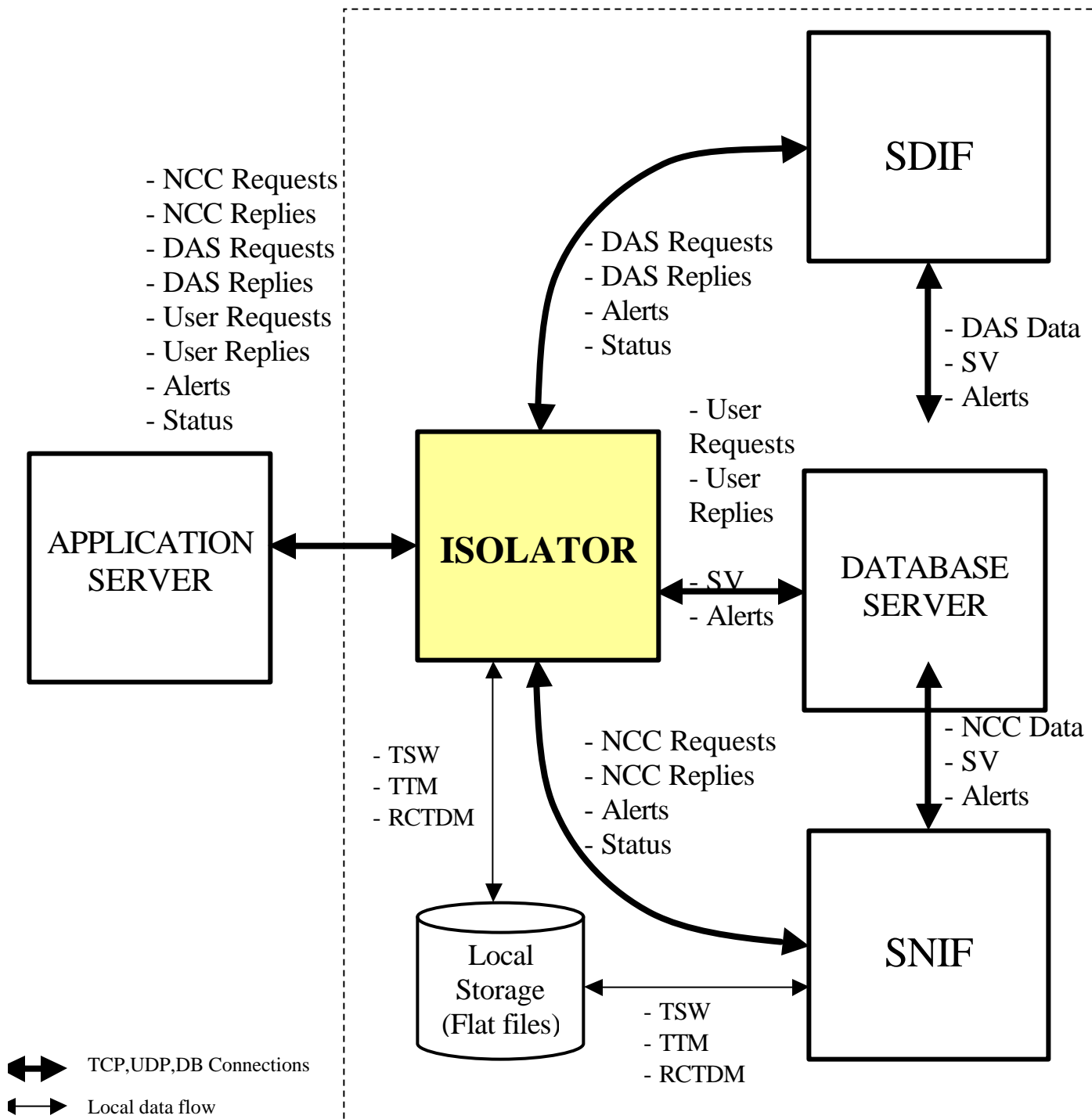


Figure 5-2 Isolator Context Diagram

As mentioned in Section 2, the SWSI Closed Server component hosts among other subsystems, one instance of SNIF and two instances of the Isolator subsystem. One instance of the Isolator connects with the Application Server running on the Open Server and the other connects with the Application Server running on the Backend Server. The functional difference between the two is one isolator supports users on the Open IONet and Internet while the other supports the users on the Closed IONet. The Isolator consists primarily of the following five major threads (see Figure 5-3):

1. Isolator Main Task (MainTask)
2. Application Server Interface (ServInterface)
3. Database Interface (DbInterface)
4. SNIF Interface (SnifInterface)
5. SDIF Interface (SdifInterface)

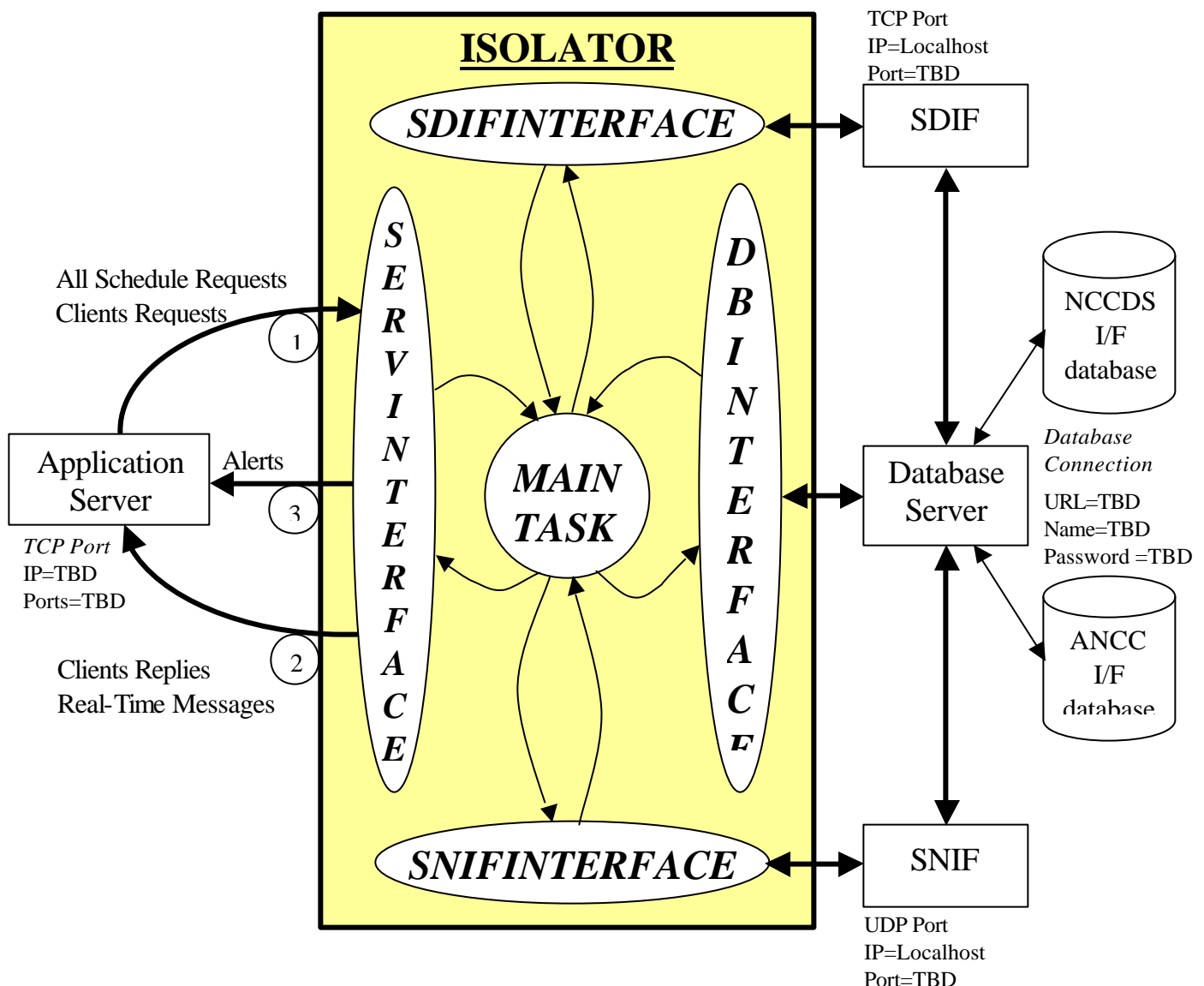


Figure 5-3 Isolator Main Threads

5.2 Isolator Main Task (MainTask):

The MainTask is the Isolator main thread that creates and starts all of the other Isolator threads. This task begins when the Isolator is started. The main purpose of this task is to manage and route all processing and I/O requests to the appropriate Isolator threads such as: ServInterface, DbInterface, SnifInterface and SdifInterface. The MainTask is the parent thread and the other four threads (just mentioned) are the subordinate threads. The MainTask will serve also as the Isolator executive task that will monitor the status and events of all the Isolator threads and queues. All of the system message logging of the Isolator subsystems is also handled by this task and are stored locally in a log file.

5.3 Application Server Interface (ServInterface):

The purpose of the ServInterface is to handle all the communications between the Application Server and the Isolator. The NCC data, DAS data and Users data as well as data stored in SWSI database pass through the ServInterface thread via the Application server on one end and the MainTask thread on the other end. The communication protocol with the Application Server is strictly TCP/IP and three ports will be assigned to route the data between the Isolator and the Application Server. The name of the Application Server host and a base port number are required to connect to the Application Server. The ServInterface will use a base port number to derive the required three port numbers to get the secure socket connections with the Application Server. There will be default host name and base port when the Isolator is started. These default parameters can be overridden by the system's environment variables where the Isolator is hosted. The ports are named respectively TP1, TP2 and TP3 and their default assignment numbers are:

- TP1 = base port number + 0
- TP2 = base port number + 1
- TP3 = base port number + 2

Each port will handle a specific set of messages, alerts, requests and responses as defined below:

5.3.1 TP1 Port

TP1 port will be used mainly to accept all the incoming messages from the Application Server. Some of TP1 messages originate from the NCC customers for delivery to the NCC and similarly some other TP1 messages originate from the DAS customers for delivery to the DAS.

5.3.1.1 SNIF TP1 Messages

The incoming TP1 messages that are bound to SNIF (see Figure 5-4) are:

- Schedule Add Request (SAR) Messages
- Alternate SAR (ASAR) Messages
- Schedule Delete Request (SDR) Messages
- Replace Request (RR) Messages
- Wait List Request (WLR) Messages
- User Reconfiguration Request (Service_Reconfiguration_Request) Messages* noted as URRM in the diagram below
- State Vector (SV) Messages
- TDRS Scheduling Window (TSW) Messages
- Multiple Ground Control Message Request (MGCMR) Messages (include the following)
 - o User Reacquisition Request (User_Reacquisition_Request) Messages
 - o Forward Link Sweep Request (Forward_Link_Sweep_Request) Messages
 - o Forward Link EIRP Reconfiguration (Forward_Link_EIRP_Reconfiguration) Messages
 - o Expanded User Frequency Uncertainty Request (Expanded_User_Frequency_Uncertainty_Request) Messages
 - o Doppler Compensation Inhibit Request (Doppler_Compensation_Inhibit_Request) Messages

The SAR, ASAR, SDR, RR, WLR, and Service_Reconfiguration_Request message data will be stored in respective tables in the SWSI database. Once the data has been successfully stored in the SWSI database, the Isolator will send Key Information to the SNIF for further processing. Similarly, the SV and TSW data will be stored locally in respective files. In this case, the Isolator will send TSW and SV file information to SNIF. The MGCMR information is not stored in the SWSI database; this information will be forwarded to the SNIF and SDIF. (see Appendix C for more details).

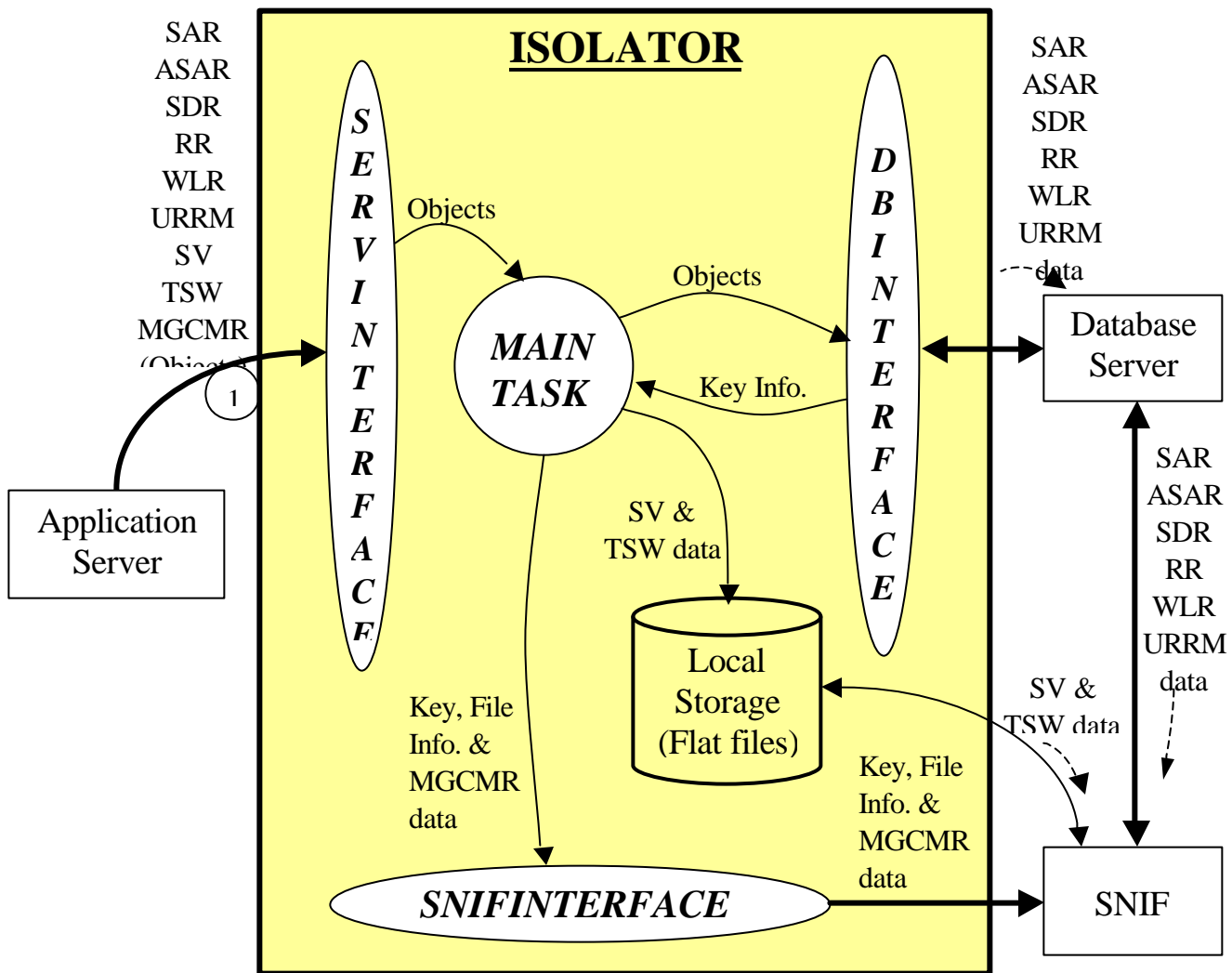


Figure 5-4 Data flow of TP1 Messages bound to SNIF/NCC

5.3.1.2 SDIF TP1 Messages

The incoming TP1 messages that are bound to DASCON (see Figure 5-5 and Figure 5-6) are:

- Resource Allocation Request (RAR) Messages
- Resource Allocation Deletion Request (RADR) Messages
- Resource Allocation Modification Request (RAMR) Messages
- Playback Request (PBKR) Messages

- Playback Deletion Request (PBKDR) Messages
- Playback Modification Request (PBKMR) Messages
- Resource Availability Request (MnemonicRequest-RAV) Messages
- Playback Search Request (MnemonicRequest-PBKS) Messages
- Planned Events Request (MnemonicRequest-USM_List) Messages
- Event Details Request (MnemonicRequest-USM_SCC_List) Messages
- Service Reconfiguration Request (Service_Reconfiguration_Request) Messages * noted as SERR in the diagram below
- Signal Reacquisition Request (User_Reacquisition_Request) Messages *noted as SigRR in the diagram below
- State Vector (SV) Messages

The RAR, RADR, RAMR, PBKR, PBKDR and PBKMR data will be stored in respective tables in the SWSI database (see Figure 5-5). Once the data has been successfully stored in the SWSI database, the Isolator will send Key Information to the SDIF. In turn, SDIF will use the key Information to read the data from the different tables in the SWSI database, format the data into XML documents and send the resulting output to the DAS.

The RAV, PBKSReq, USM_List, USM_SSC_List, SerRR, SigRR and SV messages are forwarded to the SDIF after updating their respective objects with a uniquely generated Request ID from the SWSI database; the Request ID is an Oracle sequence number. All messages transmitted from SWSI to DAS will have an incrementing Request ID. The Request ID will be kept in the SWSI database and always be updated by the Isolator or by SDIF depending of the category of the TP1 messages. For these messages just described (see Figure 5-6) the Isolator will request a new Request ID from the SWSI database and update the object prior to sending it to the SDIF. For all the other messages bound to DAS that have been stored by the Isolator in the SWSI database (see Figure 5-5), SDIF will be responsible to update the request ID prior to transmitting these messages to DAS.

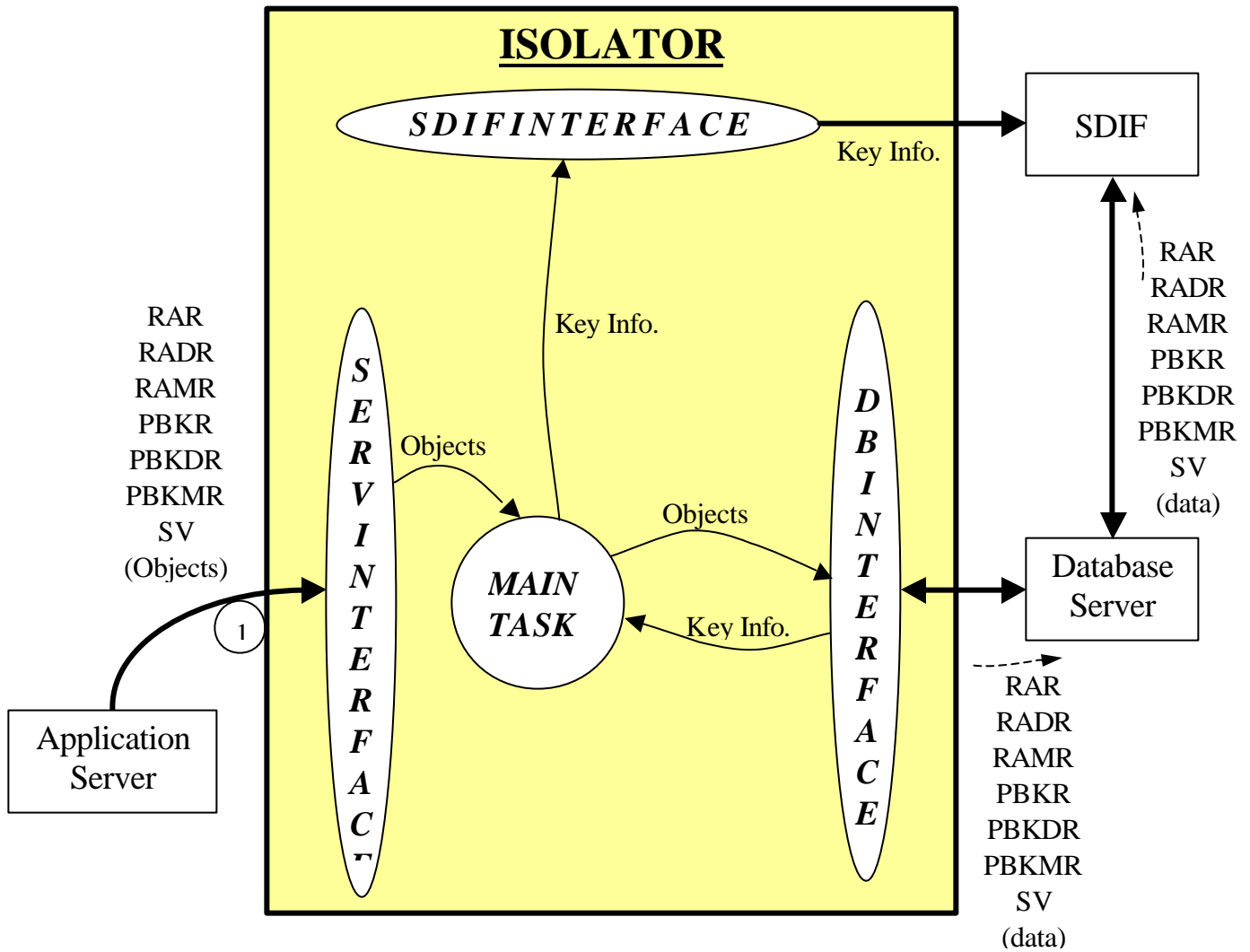


Figure 5-5 Flow of TP1 DAS Messages that get stored in SWSI database

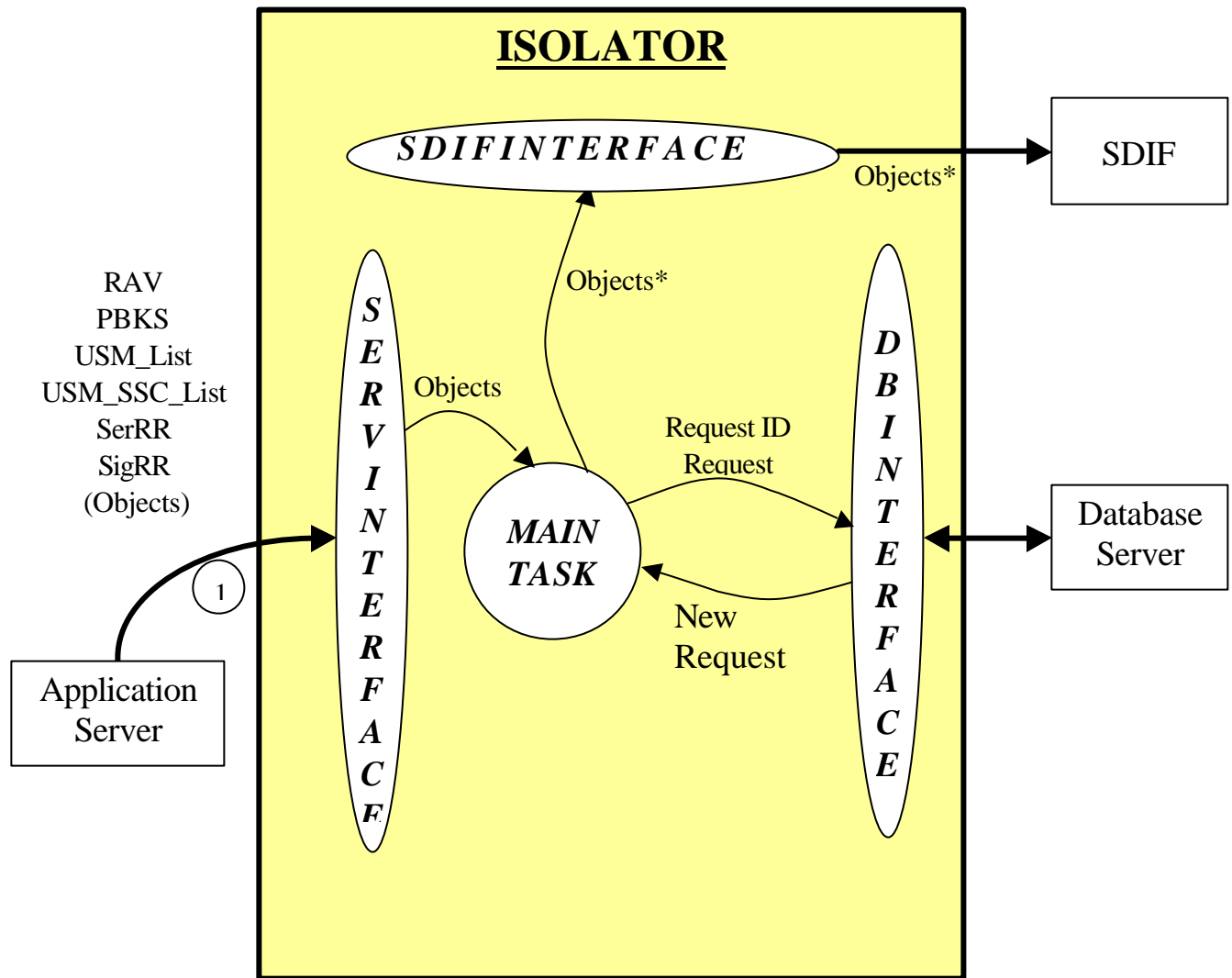


Figure 5-6 Flow of DAS Messages not stored in SWSI database

5.3.1.3 Users TP1 Messages

The last category of incoming TP1 messages are those from the SWSI or/and DAS users who are requesting information that has been stored in the SWSI or the DAS database (see Figure 5-7). These types of User Requests are:

- User Login Request (LoginObject)
- User Logout Request (LogoffObject)

- Schedule Request Summary Request (MnemonicRequest – Schedule_Request_List) *noted as SRL in the diagram
- Active Schedule Summary Request (MnemonicRequest-USM_List) *noted as USM in the diagram
- View SSC Request (MnemonicRequest-SSC)
- Modify SSC Request (ModifySSC)
- Other TBD User request (-----Req)

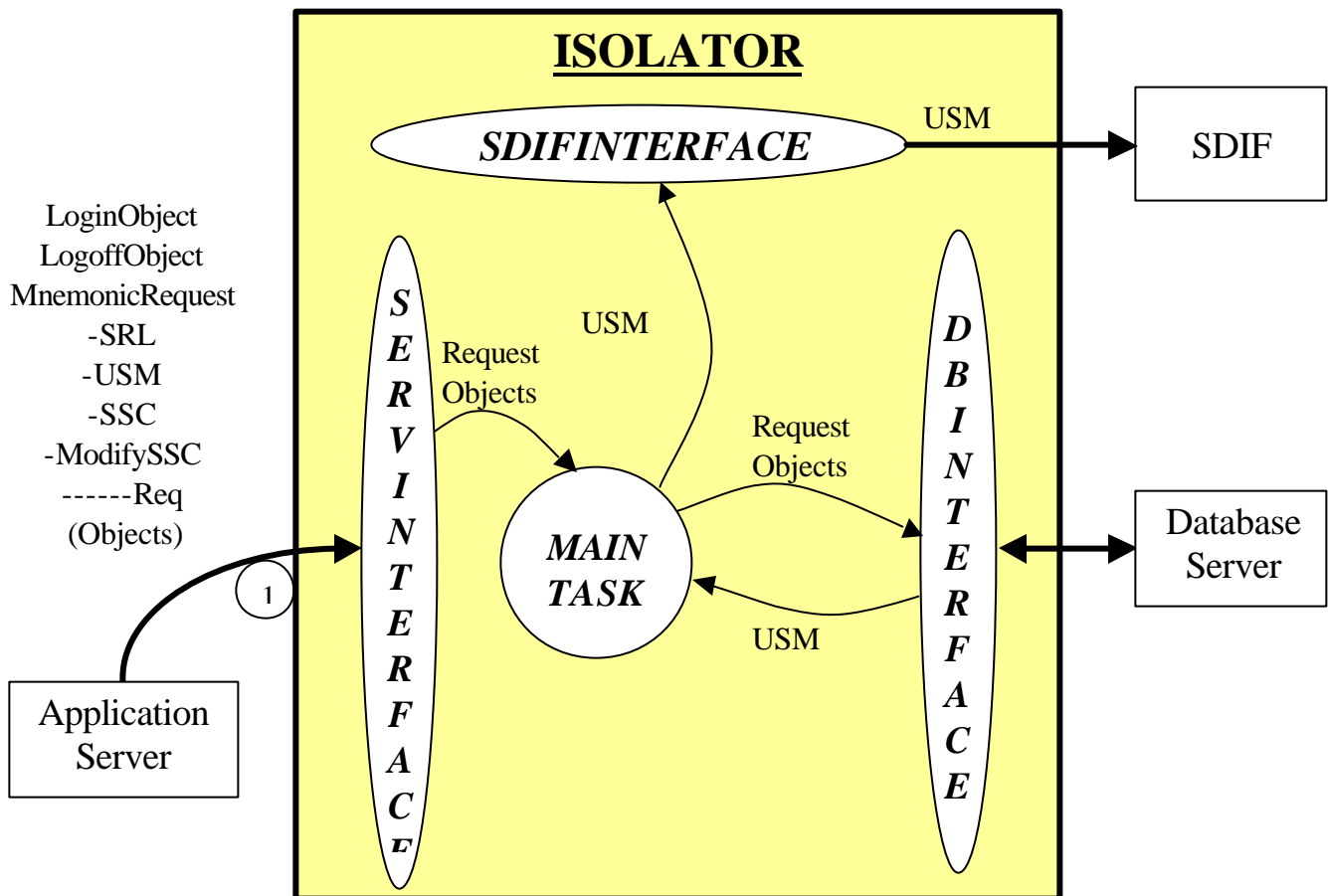


Figure 5-7 Flow of TP1 Common User Request Messages

The Active Schedule data for SWSI users is kept in the SWSI database, however, this is not the case for DAS. The Active Schedule data for DAS users is only stored in the DAS database and therefore a request must be forwarded to DASCON via the SDIF to ask for the DAS Active Schedule Summary. When the Isolator receives an Active Summary Request (USM_List) and it finds that the user requesting that data is a DAS user, the isolator will send a USM_List Request to SDIF to get the active schedules from DASCON. The Isolator will wait for the receipt of the USM_List Response from SDIF before merging the active schedule data from both NCC and DASCON. In the case where SDIF did not respond to the request after a TBD time-out, the Isolator will send to the Application Server only the active schedules from the NCC and an alert message reflecting the SDIF time-out situation. The USM_List_SSC request will work the same way as what we've just describe for the USM_List but by sending the detailed event information from both the NCC and DASCON.

5.3.2 TP2 Port

All the users' login, logout and requests messages that were received from TP1 port will generate responses that will be sent from the Isolator to the Application Server via TP2 port. TP2 will also be used to transmit critical real time messages such as UPD messages to the Application Server. The following figure (5-8) describes the flow of TP2 Messages.

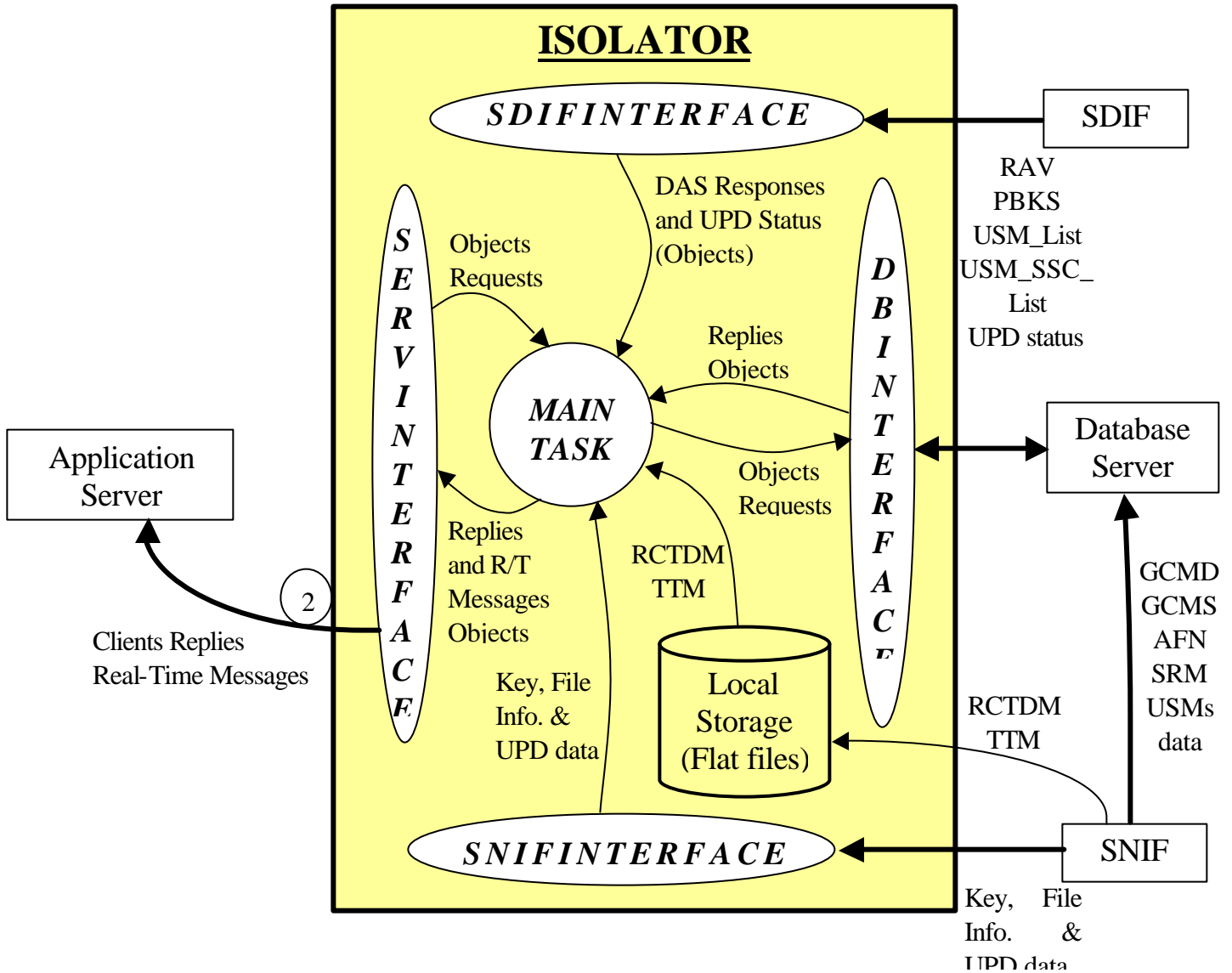


Figure 5-8 TP2 Data flow Messages

User Request for data from the application server will be wrapped into MnemonicRequest objects. Isolator responses to the application server will be wrapped into MnemonicData objects. The mnemonic name given in these objects would indicate the type of data request/reply. A detail description of the MnemonicRequest and MnemonicData objects with all the supported messages is given in the SWSI Common Objects section in Appendix A.

5.3.3 TP3 Port

The third port TP3 will be used only to transmit the alerts messages to the Application Server. All the alert messages from the Isolator, SNIF and SDIF are stored in the SWSI database. SNIF will be responsible to store its generated alert messages directly into the SWSI database and at the same time

forward them to the Isolator for further transmission to the Application Server. However, the Isolator will store SDIF generated alerts as well as Isolator generated alerts into the SWSI database. Figure 5-8 describes the data flow of the Alert Messages.

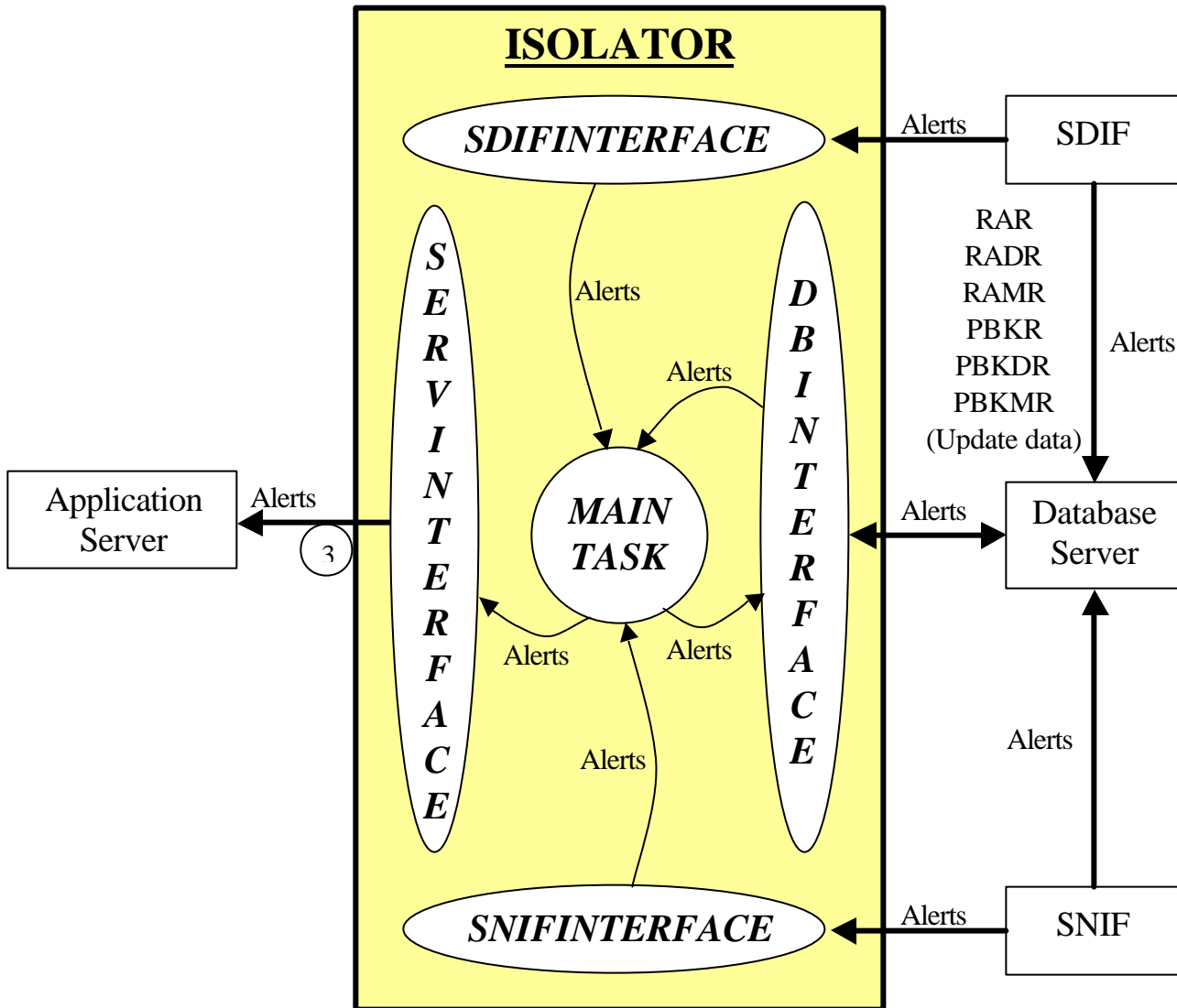


Figure 5-9 TP3 Data Flow of the Alerts Messages

Appendix C describes in detail the format of the alert messages between SNIF and the Isolator. SNIF alert messages will be converted by the Isolator into alert objects conforming to the Alert class as defined in the SWSI common classes and then transmitted as serialized Java object to the Application server.

The ServInterface will be responsible to get the name and base port number of the Application Server, to initiate its communication threads and to monitor their status. The ServInterface will create 3 subordinates threads each responsible for processing the data of the 3 ports naming TP1Thread for port 1, TP2Thread for port 2 and TP3Thread for port 3. All the threads will perform their respective I/O processing independently and will exchange the data with the other Isolator threads/objects via their parent task ServInterface. In turn, the ServInterface task will exchange data with other objects via its parent task MainTask, thus following a top down hierarchy. All the threads are required for the normal operation of the Isolator, which also means that all 3 sockets connections must be always up, and running with the Application Server. When the connection of one of the port is lost, ServInterface task will turn off the other active connections and will restart itself for new connections with the Application Server. The log file will be updated with connection events that will indicate the time the connections were made or lost. The exchange of data between the Isolator and the Application Server are done in serialized objects and are described in Appendix A.

5.4 Database Interface (DbInterface):

The purpose of the Database Interface (DbInterface) is to store, retrieve and update SWSI data found in the SWSI database. Data sent from the Application Server will be stored in the appropriate SWSI database tables and requested data will be retrieved and sent to the ServInterface task for transmission to the Application Server. The DbInterface task will connect to the SWSI database hosted on the same local machine as the Isolator. The DbInterface requires the name of the Oracle data base driver, the URL of the SWSI database instance, an authorized account name and password to the database. Once connected, DbInterface will make JDBC calls to PL/SQL stored procedures for data storage, retrieval and updates.

It is important to note that the Isolator, SDIF and SNIF will be storing data to the SWSI database. The user data coming from the Application Server to the Isolator along with alert messages generated by the Isolator are stored by the DbInterface task. All updates to the SWSI database by the Isolator, SDIF and SNIF are synchronized by Oracle to avoid any loss or corruption of data. Both of the Isolators and SNIF will be polling the database at a low priority to detect any new data stored by the other subsystem.

5.5 SNIF Interface (SnifInterface):

The purpose of the SnifInterface is to handle all the communications between the SNIF and the Isolator. The communication protocol with SNIF is strictly UDP and only one port will be assigned to route the data between the Isolator and SNIF. The name of the host computer where SNIF is located and a communication port number are both required exchanging data between both subsystems. As we've

mentioned before, SNIF and the Isolator will be located on the same platform; therefore, the default name of the host platform for both will be referred as “localhost”. The default port number is set to 31416; the default port number can be overridden by changing the appropriate system environment variable.

5.6 SDIF Interface (SdifInterface):

The purpose of the SdifInterface is to handle all the communications between the SDIF and the Isolator itself. The communication protocol with SDIF is TCP/IP and only one connection port will be assigned to route the data between the Isolator and SDIF. The Isolator will be the client side and SDIF will be the Server Side. Similarly to SNIF, SDIF will also be collocated with the Isolator and therefore, the default name of the host platform will be referred as “localhost”. The default port number to connect to SDIF is set to 31417 but can also be changed via the setting of the system environment variables.

5.7 Logging

The Isolator subsystem logs all the OS or database related errors in a log file. System messages are also logged and are time-tagged along with the source of the thread from where they were issued. The log file(s) can be viewed at anytime by SWSI developers for analysis and/or debugging. All the alerts generated by Isolator are logged and then stored in the ALERT_Message database table.

Section 6. SWSI-NCCDS Interface Design

6.1 Overview

The SWSI-NCCDS Interface (SNIF) performs all electronic message-based communication with the operational NCCDS and with the ANCC for performing Engineering Interface (EIF) testing. The SNIF establishes and maintains all the TCP connections required for implementing the full customer message interface as defined in the NCCDS/MOC ICD for full support customers. Separate sets of connections are maintained for each group of missions as defined in the SWSI database. The SNIF is also responsible for maintaining the active schedule in the SWSI database based on the Schedule Result Message (SRM) and User Schedule Message (USM) responses received from the NCCDS. The following sections describe the operating environment of the SNIF and the detailed design.

6.2 Operating Environment

SNIF is a multi-threaded "C" application that executes as a single Unix process under Solaris 7. SNIF uses the POSIX thread Application Programming Interface (API), or pthreads, a standardized threading implementation that allows asynchronous execution of concurrent tasks within a single application. The primary benefit of threading to the SNIF application is to allow the efficient control of multiple TCP socket connections without having to resort to a complicated mechanism of non-blocking I/O and polling. A separate POSIX thread controls each connection. If a socket read is blocked because data is unavailable, the thread is suspended to allow execution of other threads within the SNIF application.

Synchronization or communication between threads is provided by a mutual exclusion mechanism called the mutex. The mutex allows for locking of data that is shared between threads.

The primary mechanism used by the SNIF for communication between threads is the message queue. Since pthreads doesn't provide queueing, a custom package is used that uses mutexes for locking queue data structures. The SNIF queue model is meant to roughly imitate a similar capability provided in the old Ready Systems VRTX32 real-time embedded OS kernel. In general, it works by allowing threads to "post" a message buffer to another thread's queue. A thread retrieves messages from its own queue by "pending", which suspends the thread until a message arrives, or by "accepting", which does not suspend the thread if its input queue is empty.

6.3 Detailed Design

The context diagram for the SNIF is shown in Figure 6-1. Communications with the Isolators is according to the protocol described in Appendix C. Each Isolator will listen on a separate User Datagram Protocol (UDP) port, while the SNIF will receive messages on a single UDP port. All outbound messages will be sent to both Isolators, since the SNIF will have no knowledge about which

clients should receive a particular message. The Application Servers will be responsible for examining the messages to determine which open or closed client(s) should receive the messages.

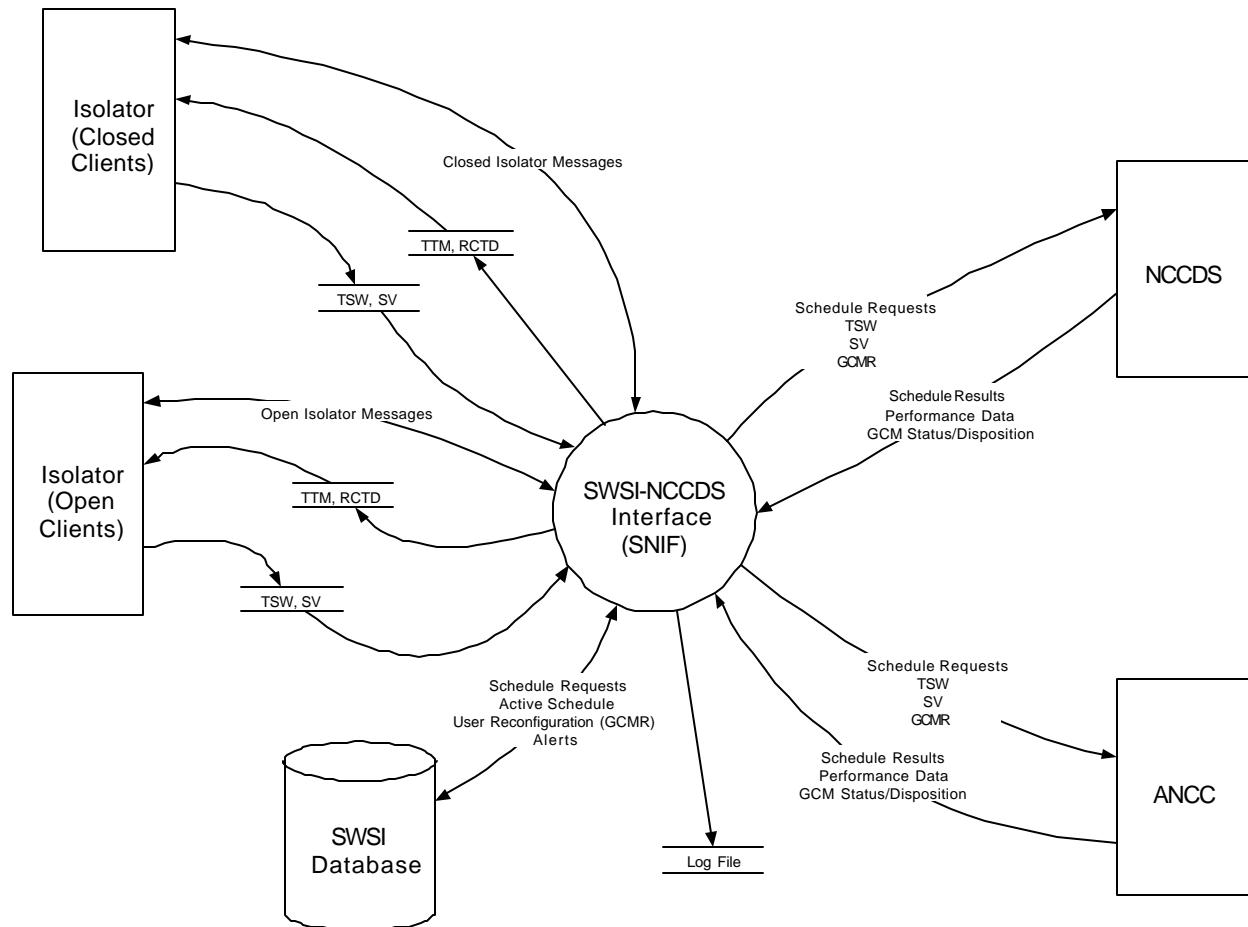


Figure 6-1 SNIF Context Diagram

The SNIF communicates with both the operational NCCDS and the ANCC. A separate set of connections is used for each facility.

The top-level data flow diagram is shown in Figure 6-2. Each bubble represents a functional process that eventually decomposes into a number of primitive threads. The C program "main" is used to initialize the software and create the appropriate threads. After "main" completes execution, each thread runs independently. Communication between threads is primarily through message queues. Queues provide a pipeline for messages to be passed between threads. All queue operations are performed by functions provided within the SNIF application.

Alert messages are shown in the context diagram but are not shown in any of the lower level data flow diagrams. Any SNIF thread is capable of generating an alert to be sent to the Isolators and stored in the Database. The alerts are not shown so as to simplify the diagrams.

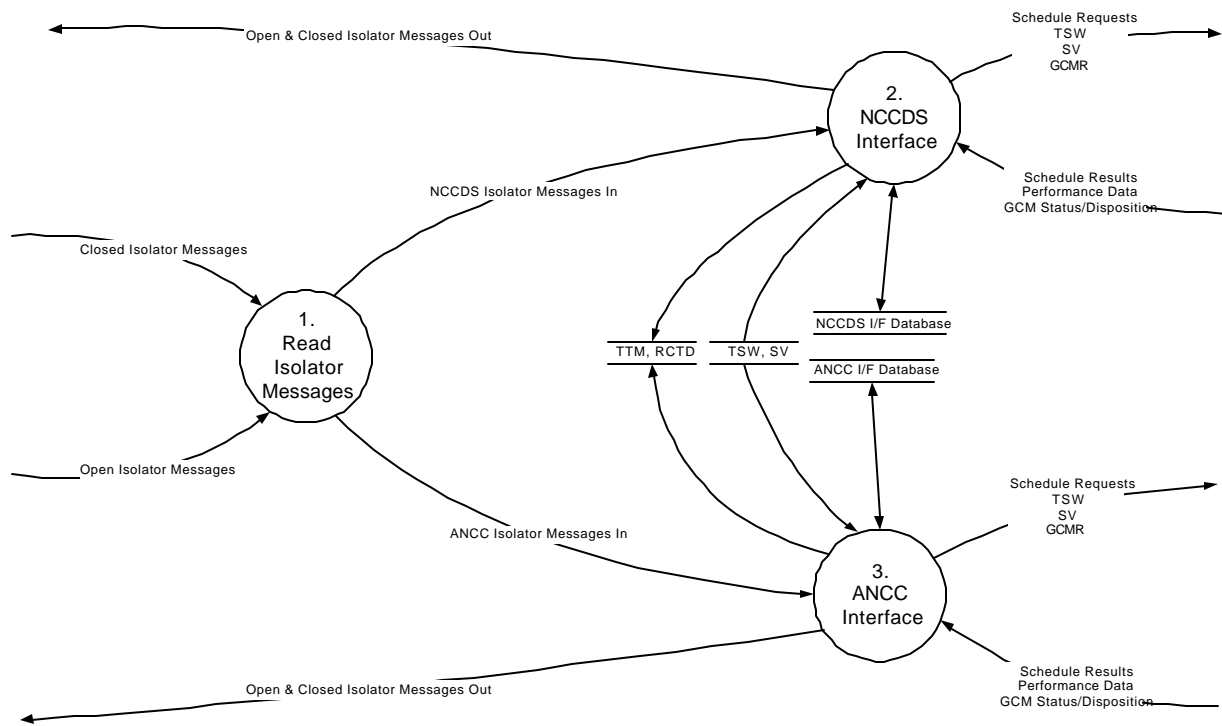


Figure 6-2 SNIF Level 0 Data Flow Diagram

A detailed description of each process and primitive thread is given below. A separate functional process is provided for each of the NCC interfaces (NCCDS and ANCC). Data flow diagrams for these processes are given in Figures 6-3 and 6-4. The processes are identical with the exception that they each access a different database instance and communicate with a different NCC system. Only a description of the NCCDS Interface is provided.

6.3.1 Read Isolator Messages

Read Isolator Messages is a thread that is primarily responsible for routing incoming Isolator messages. The Read Isolator Messages thread listens for messages from both Isolators on a single UDP port. The messages are routed to the appropriate NCC interface process depending on the routing indicator in the message header.

6.3.2 NCCDS Interface

The NCCDS Interface process controls all communication with the NCCDS. Incoming Isolator messages are received through the Read Isolator Messages thread. Outgoing Isolator messages are

sent directly to both the open and closed Isolators. A data flow diagram for the NCCDS Interface process is given in Figure 6-3.

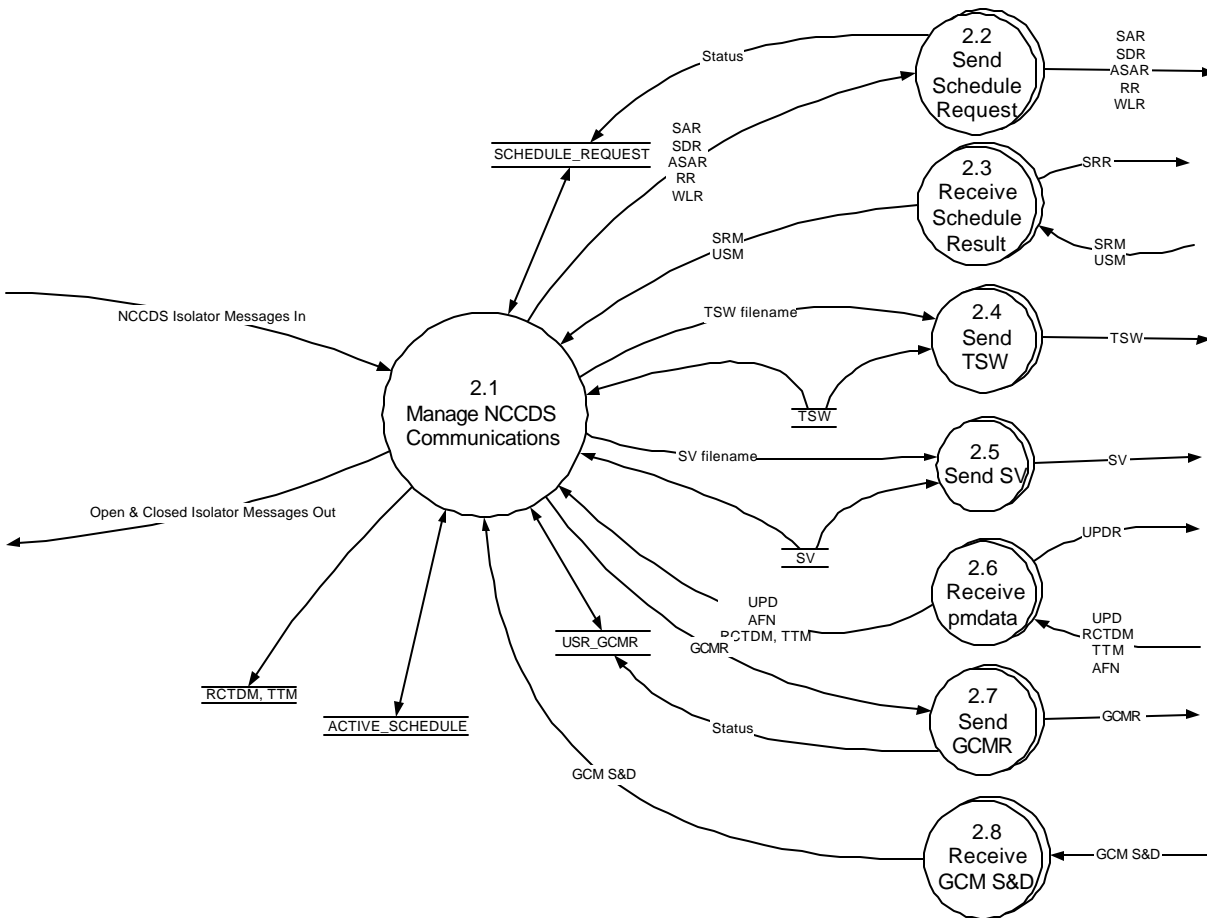


Figure 6-3 NCCDS Interface Data Flow Diagram

Manage NCCDS Communications is the primary thread responsible for processing NCCDS messages. The remaining threads are used to control the NCCDS connections as defined in Table 4-3 of the NCCDS/MOC ICD. A separate thread is assigned for each service, with two threads required for the reconfig (GCMR) service because data flow is bi-directional. Only single threads are needed for the schStatus and pmData services because the Schedule Result Request (SRR) and User Performance Data Request (UPDR) messages are treated simply as initialization messages to be transmitted when the connection is first established. A separate instance of each thread is used for each NCCDS connection.

The connection control threads serve two functions. The first function is to establish and maintain connections to the appropriate NCCDS services according to the configuration described in the SCHEDULE_CONNECTION and REALTIME_CONNECTION Database tables. These

configurations are read at SNIF startup. Changes in the configuration will require a restart of the SNIF application.

The connections for the schStatus, pmData, and reconfig services will be permanent connections. The schReq, tswStore, and acqStore connections will be established only when there are requests to be transmitted. The connections will be timed out after periods of inactivity as defined by the Database. The other function of these threads is to read or write messages in eXternal Data Representation (XDR) format on the connections being controlled. The messages exchanged with the Manage NCCDS Communications thread are in the raw formats as defined in the ICD.

6.3.2.1 Manage NCCDS Communications

Manage NCCDS Communications is a single thread that performs the majority of the processing of messages passing through the SNIF. For messages outbound to the NCCDS, the Manage NCCDS Communications thread receives information about a message or request from an Isolator message and database entry and constructs the NCCDS message in the appropriate format. For messages inbound from the NCCDS, the Manage NCCDS Communications thread interprets the message for immediate processing as in the case of SRMs and USMs, reformats the message for transfer to the client as in the case of User Performance Data (UPD), or generates a SWSI-formatted alert as in the case of Ground Control Message (GCM) Status and Disposition (S&D) and Acquisition Failure Notification (AFN).

The Manage NCCDS Communications thread will poll the Database periodically for requests that have been saved but not transmitted. This is to compensate against the loss of an Isolator UDP message that indicates storage of a request message. The file system directories holding TDRSS Scheduling Window (TSW) and State Vector (SV) messages will be similarly polled so that messages aren't held indefinitely by SNIF due to lost Isolator messages.

Following is a description of how each NCCDS message is processed by the Manage NCCDS Communications thread.

6.3.2.1.1 Schedule Request

Schedule request messages (SAR, SDR, ASAR, RR, WLR) are received by the SNIF from the Isolator in the form of a requestId key referencing a record in the SCHEDULE_REQUEST table. The NCCDS message is constructed from this information. The message is queued to the appropriate Send Schedule Request thread and the status of the request is changed to QUEUED.

6.3.2.1.2 Schedule Result Message

Schedule Result Messages (SRMs) are received from the Receive Schedule Result thread. The status of the referenced request is updated in the SCHEDULE_REQUEST table in the database based on the result and explanation codes received in the SRM. If the codes indicate a deletion and the event had been previously scheduled, then the event is deleted from the ACTIVE_SCHEDULE table. In the case of all SRMs received, an alert is sent to the Isolators for display on the client(s).

6.3.2.1.3 User Schedule Message

User Schedule Messages (USMs) are received from the Receive Schedule Result thread. The USM is used to add an event to the ACTIVE_SCHEDULE and associated database tables. If an event is stored in ACTIVE_SCHEDULE with the same event ID, then that event is overwritten with the information from the new USM. An alert is sent to the Isolators for each USM received.

6.3.2.1.4 TDRS Scheduling Window Message

TDRS Scheduling Window (TSW) messages are received by the SNIF from the Isolator in the form of a filename. The TSW is validated to ensure that the SUPIDEN matches the SIC from the Isolator message header. The filename information is then passed through to the appropriate Send TSW thread as determined by the Isolator message SIC.

6.3.2.1.5 State Vector

State Vector (SV) messages are received by the SNIF from the Isolator in the form of a filename. The SV is validated to ensure that the SIC in the vector matches the SIC from the Isolator message header. The filename information is then passed through to the appropriate Send SV thread as determined by the Isolator message SIC.

6.3.2.1.6 User Performance Data

User Performance Data (UPD) messages are received from the Receive pmdata thread. The UPDs are parsed and reformatted from binary data into name-value pairs. The name-value pairs are stored in PD messages that are sent to the Isolators. The PD message types are as described in the UPD database table.

6.3.2.1.7 Acquisition Failure Notification

The Acquisition Failure Notification (AFN) is received from the Receive pmdata thread and is sent to the Isolators as an alert.

6.3.2.1.8 Return Channel Time Delay Message

The Return Channel Time Delay Message (RCTDM) is received from the Receive pmdata thread. The message is stored in raw form in a file. The filename information is sent to the Isolators.

6.3.2.1.9 Time Transfer Message

The Time Transfer Message (TTM) is received from the Receive pmdata thread. The message is stored in raw form in a file. The filename information is sent to the Isolators.

6.3.2.1.10 Ground Control Message Request

User Reconfiguration Request messages, which are a type 98/04 Ground Control Message Request (GCMR), are received by the SNIF from the Isolator in the form of a msgId key referencing a record in the USR_GCMR table. The NCCDS message is constructed from this information. The message is queued to the appropriate Send GCMR thread and the status of the request is changed to QUEUED.

The remaining GCMR message types are received from the Isolator as messages described by name-value pairs. The NCCDS message is constructed from the name-value pairs. The message is then queued to the appropriate Send GCMR thread.

6.3.2.1.11 Ground Control Message Status and Disposition

Ground Control Message (GCM) status and disposition messages are received from the GCM S&D thread. The status of the referenced request is updated in the USR_GCMR table. If the referenced GCMR is a User Reconfiguration Request and the GCMR was accepted, then the USR_GCMR table is updated to reflect the new parameter settings. An alert is sent to the Isolators for each GCM status and disposition received.

6.3.2.2 Send Schedule Request

The Send Schedule Request thread sends messages to the NCCDS through the schReq service. A connection is established only when there is a request to be transmitted. The connection is closed after a period of inactivity. Schedule request messages (SAR, SDR, ASAR, RR, WLR) are received from the Manage NCCDS Communications thread. The User ID and Password are inserted into the message. If a connection is successfully established and the message is transmitted, then the status of the request in SCHEDULE_REQUEST is changed to TRANSMITTED and an alert is sent to the Isolator indicating a successful transmission.

6.3.2.3 Receive Schedule Result

The Receive Schedule Result thread receives messages from the NCCDS through the schStatus service. A permanent connection is maintained on this service. When first established, a Schedule Result Request (SRR) message is constructed from configuration information in the database and is sent to the NCCDS. Schedule Result Messages (SRMs) and User Schedule Messages (USMs) received on this connection are sent in raw form to the Manage NCCDS Communications thread.

6.3.2.4 Send TDRSS Scheduling Window

The Send TDRSS Scheduling Window (TSW) thread sends messages to the NCCDS through the tswStore service. A connection is established only when there is a message to be transmitted. The connection is closed after a period of inactivity. TSW messages are received from the Manage NCCDS Communications thread in the form of a filename. The User ID and Password are inserted into the message. If a connection is successfully established and the message is transmitted, then the file is moved to an archive directory and an alert is sent to the Isolator indicating a successful transmission.

6.3.2.5 Send State Vector

The Send State Vector (SV) thread sends messages to the NCCDS through the acqStore service. A connection is established only when there is a message to be transmitted. The connection is closed after a period of inactivity. SV messages are received from the Manage NCCDS Communications thread in the form of a filename. If a connection is successfully established and the message is transmitted, then the file is moved to an archive directory and an alert is sent to the Isolator indicating a successful transmission.

6.3.2.6 Receive pmdata

The Receive pmdata thread receives messages from the NCCDS through the pmData service. A permanent connection is maintained on this service. When first established User Performance Data Request (UPDR) messages are constructed from configuration information in the database and are sent to the NCCDS. A separate UPDR is sent for each SUPIDEN supported on that connection. All UPDRs sent will be to enable UPD transmission. UPDs, Acquisition Failure Notification (AFN), Return Channel Time Delay Messages (RCTDMs), and Time Transfer Messages (TTMs) received on this connection are sent in raw form to the Manage NCCDS Communications thread.

6.3.2.7 Send Ground Control Message Request

The Send Ground Control Message Request (GCMR) thread sends messages to the NCCDS through the reconfig service. A permanent connection is maintained on this service. GCMRs are received from the Manage NCCDS Communications thread. The User ID and Password are inserted into the message. If the message is successfully transmitted, then an alert is sent to the Isolator indicating a successful transmission. For type 98/04 GCMRs the status of the request in USR_GCMR is changed to TRANSMITTED.

6.3.2.8 Receive Ground Control Message Status and Disposition

The Receive Ground Control Message Status and Disposition (GCM S&D) thread receives message from the NCCDS through the reconfig service. GCM S&D messages received on this connection are sent in raw form to the Manage NCCDS Communications thread.

6.3.2 Logging and Delogging

The SNIF logs all formatted messages exchanged with the NCCDS, as well as significant events and errors such as connection establishment and loss. The logs will use the NCCDS Central Delogger (NCD) format. The NCCDS Protocol Gateway (NPG) delogger will be used to delog and display previously logged data.

The SNIF will provide an additional level of logging to use in debugging application or system problems. This logging will be under the control of a debug flag that is set when the application is invoked. All Isolator messages will be logged, as well as additional actions such as updates to Database tables,

updates to SNIF global tables, and queue message posts and pends. Debug output will be written to a text file for viewing and editing by standard Unix utilities.

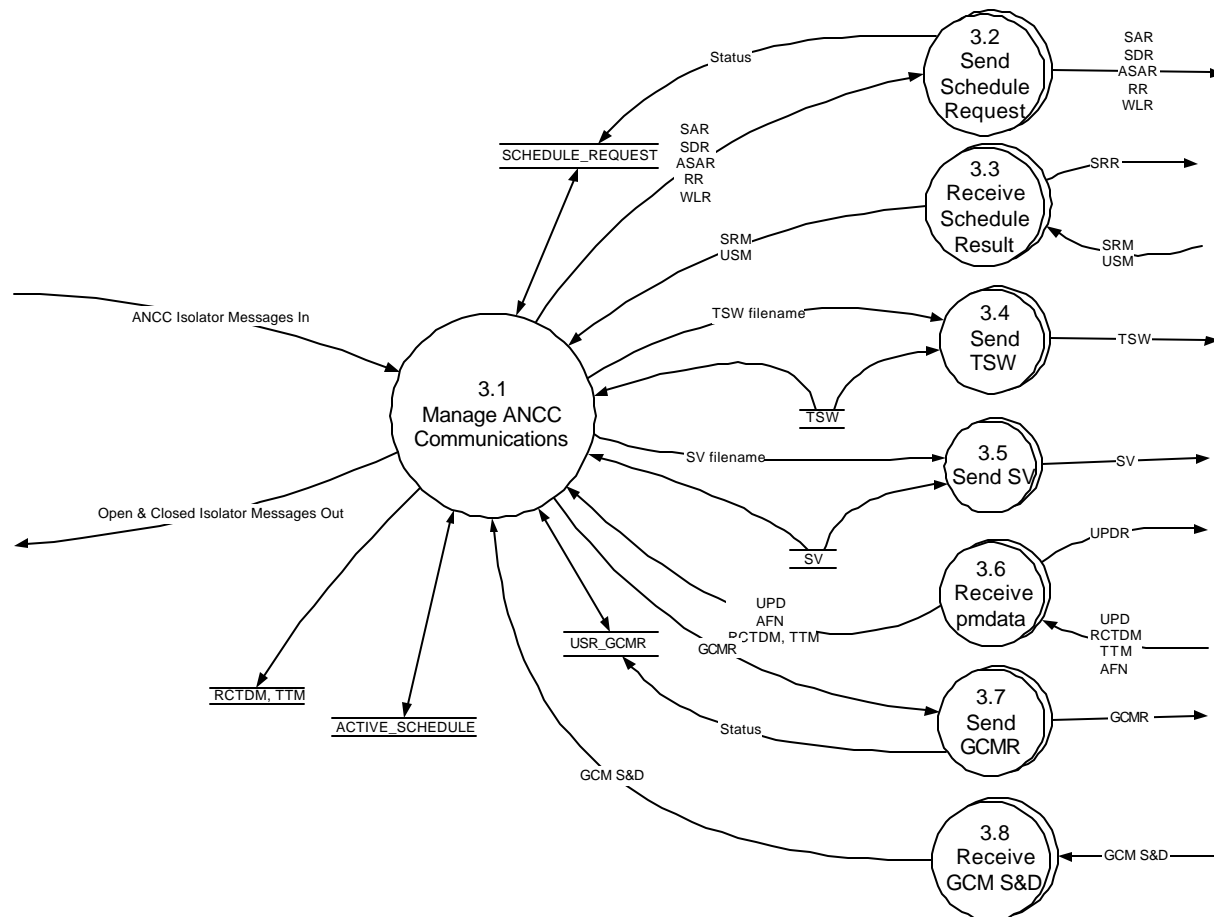


Figure 6-4 ANCC Interface Data Flow Diagram

7. SWSI-DAS Interface Design

7.1 Overview

The SWSI-DASCON Interface (SDIF) performs all electronic message-based XML-based communication with DASCON (See Figure 7-1). The SDIF establishes and maintains a TCP connection required for implementation of XML messaging interface with DASCON as well as SDIF maintains another TCP connection with Isolator for passing messages from/to Isolator/DASCON. The SDIF is responsible for converting serialized objects coming from Isolator into XML structures, which are sent to DASCON, and converting XML structures coming from DASCON into serialized Java objects for sending them to Isolator. The SDIF is also responsible for logging Alert messages and other messages considered to be "Alert" messages into database. The following sections describe the detailed design of the SDIF.

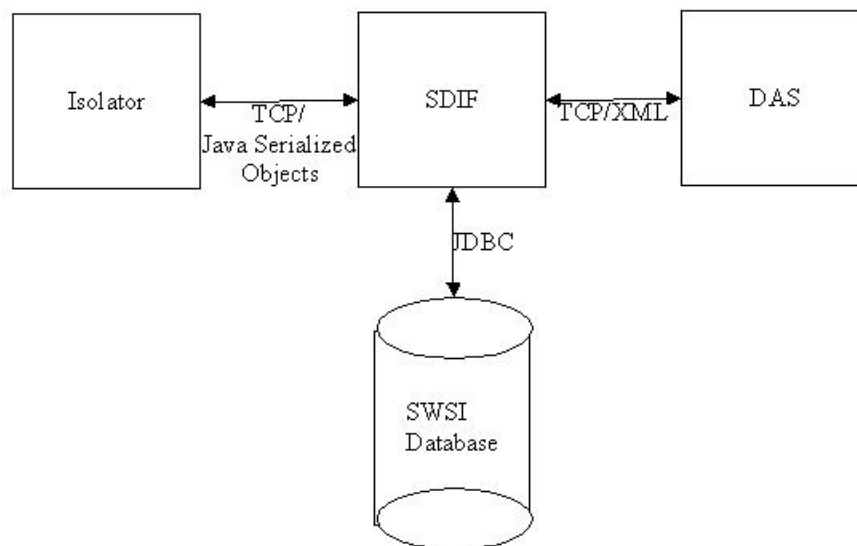


Figure 7-1 SDIF Context Diagram

7.2 SDIF Functionality

The SDIF's functionality can be broken into three main pieces:

- Handling outgoing traffic – Isolator to DASCON Interface
- Handling incoming traffic – DASCON to Isolator Interface
- Contacting database for logging and periodic message retransmission.

The following sections will describe in detail listed above tasks.

7.2.1 Isolator to DASCON Interface

The SDIF keeps a TCP connection to Isolator in order to exchange messages back and forth. On this connection Isolator is delivering messages as serialized Java objects to the SDIF. When SDIF receives a message it de-serializes it into respective Java object and decides what action should be taken according to a message. If a message receives wrapped into mnemonic message object then the SDIF extracts actual message from it.

7.2.1.1 Detailed design

The messages flowing from Isolator to DASCON through SDIF can be broken down into three categories with the appropriate actions taken (See Figure 7-2):

- Service Allocation Messages – the message is wrapped into mnemonic object. The message itself contains specification of primary key by which the message can be found in the database. The SDIF finds it in the database and converts it into respective XML instance. The XML instance is run through validating XML parser to make sure the XML is valid according to SWSI-DASCON XML Schema definition. If XML instance is invalid, then a SWSI alert message is generated back to Isolator and logged to database. Otherwise transmission of the message to DASCON is tried. If transmission is successful, then the SDIF sets the database field on the record for this message to transmitted status. If transmission fails, then SDIF's retransmission thread will be responsible for trying to retransmit the message every repeat interval (configurable parameter).
- State Vectors – the message is wrapped into mnemonic object. The message itself contains specification of primary key by which the state vector can be found in the database. SDIF fetches the proper state vector (only DASCON needed fields) from the database and constructs it into appropriate XML instance. The XML instance is run through validating XML parser to make sure the XML is valid according to SWSI-DASCON XML Schema definition. If XML instance is invalid, then a SWSI alert message is generated back to Isolator and logged to database. Otherwise transmission of the message to DASCON is tried. If transmission is successful, then the SDIF sets the database field on the record for this message to transmitted

status. If transmission fails, then SDIF's retransmission thread will be responsible for trying to retransmit the message every repeat interval (configurable parameter).

- All other messages are queued. As each message is de-queued from the queue, it is converted to the appropriate instance of XML. The XML instance is run through validating XML parser to make sure the XML is valid according to SWSI-DASCON XML Schema definition. If XML instance is invalid, then a SWSI alert message is generated back to Isolator and logged to database. Otherwise transmission of the message to DASCON is tried. If transmission fails, then a SWSI alert message is generated to Isolator indicating that there was problem with transmission.

7.2.2 DASCON to Isolator Interface

The SDIF keeps a TCP connection to DASCON in order to exchange messages back and forth. On this connection DASCON delivers messages to SDIF as XML Instances conforming to SWSI-DASCON XML Schema Definition. When SDIF receives an XML Instance formatted message, it runs it through XML validating parser using SWSI-DASCON XML Schema Definition. If the instances passes validation test, then further actions are taken as described in the next section, otherwise SWSI Alert is generated and send to Isolator as well as logged to the database.

7.2.2.1 Detailed Design

The messages flowing from DASCON to Isolator through SDIF can be broken down into three categories with the appropriate actions taken (See Figure 7-3):

- "Pure" Alert Messages – the SDIF logs such messages to database, converts them from XML to SWSI Alert message and transfers it to Isolator.
- Messages that are alerts according to the matrix - the SDIF logs such messages to database, converts such messages from XML to SWSI Alert messages and sends them to Isolator.
- All other messages are converted from XML to respective SWSI message type and transferred to Isolator

7.2.3 Retransmission Thread

As part of SDIF's execution, separate Thread (Retransmission Thread) is launched to handle the task of re-transmitting those messages that failed to transmit at the original request time (when a message first received from Isolator). The Retransmission Thread queries database every repeat interval (configurable parameter) for records with saved status (that is saved but not transmitted messages). If such records found, then Retransmission Thread constructs an appropriate XML instance for each message and tries to retransmit it to DASCON. If message expired and re transmission still failed, then the Retransmission Thread generates SWSI Alert message back to Isolator indicating that transmission failed and message expired (See Figure 7-4).

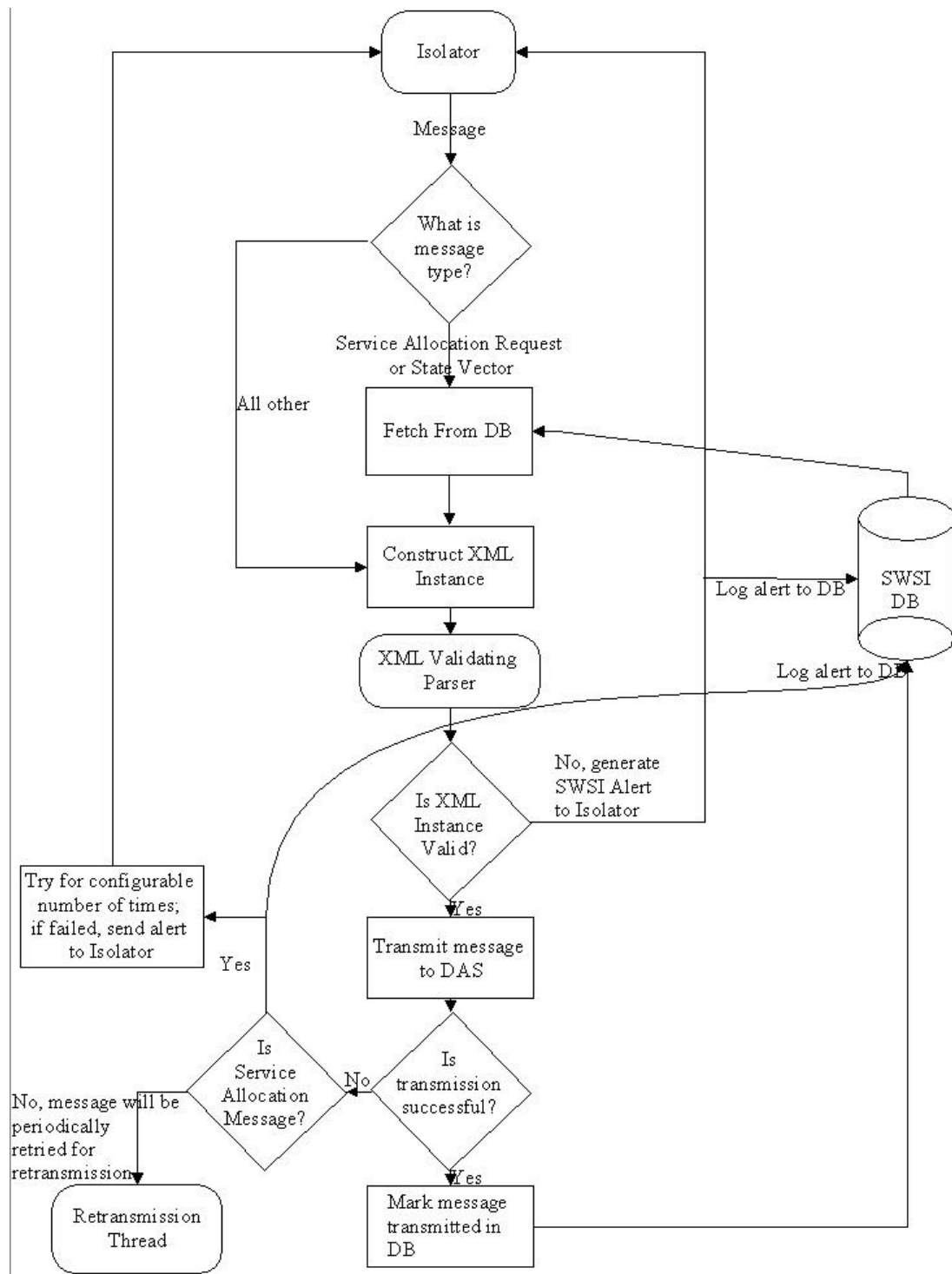


Figure 7-2 Control Logic Overview

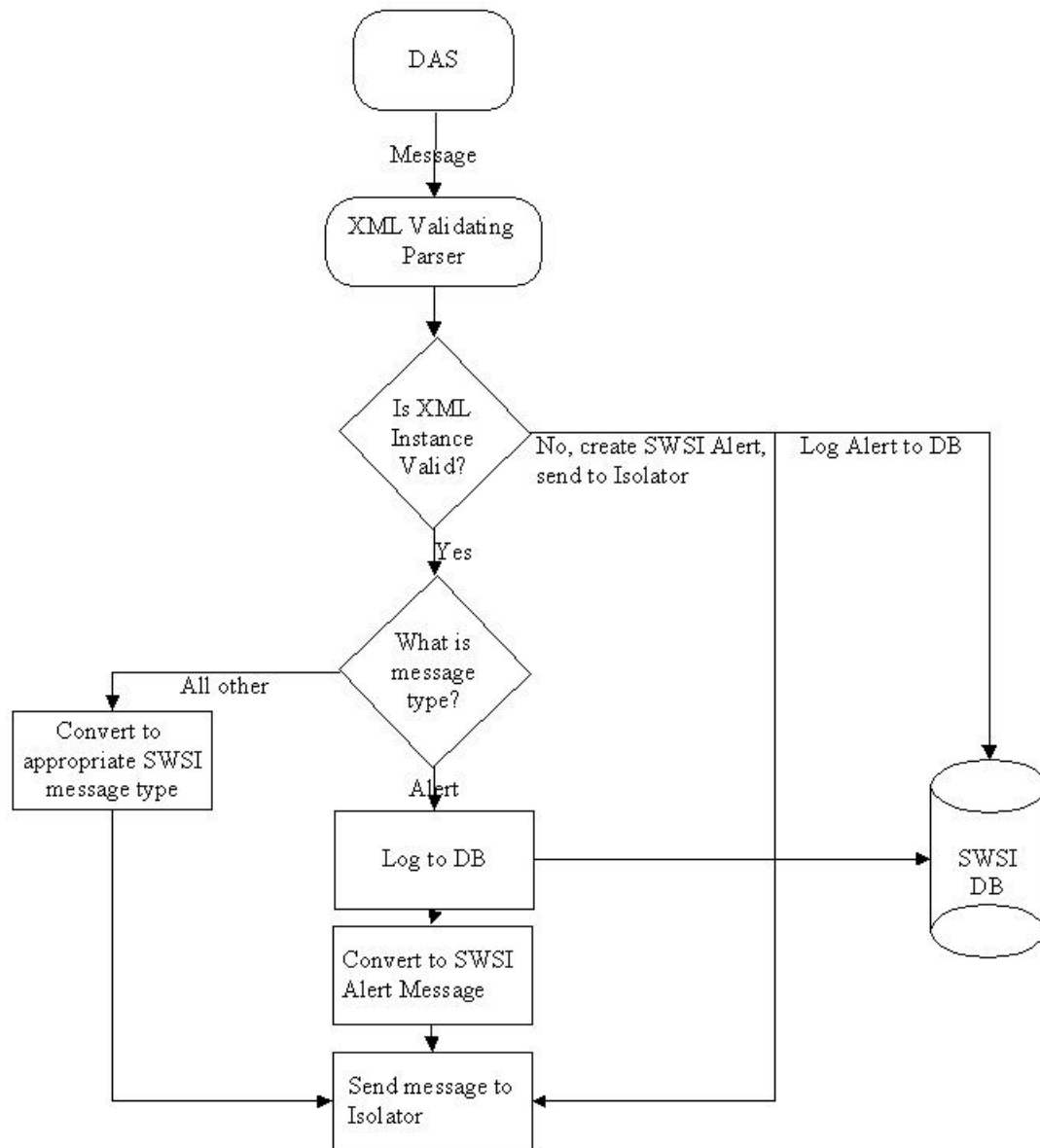


Figure 7-3 Detailed Control Logic

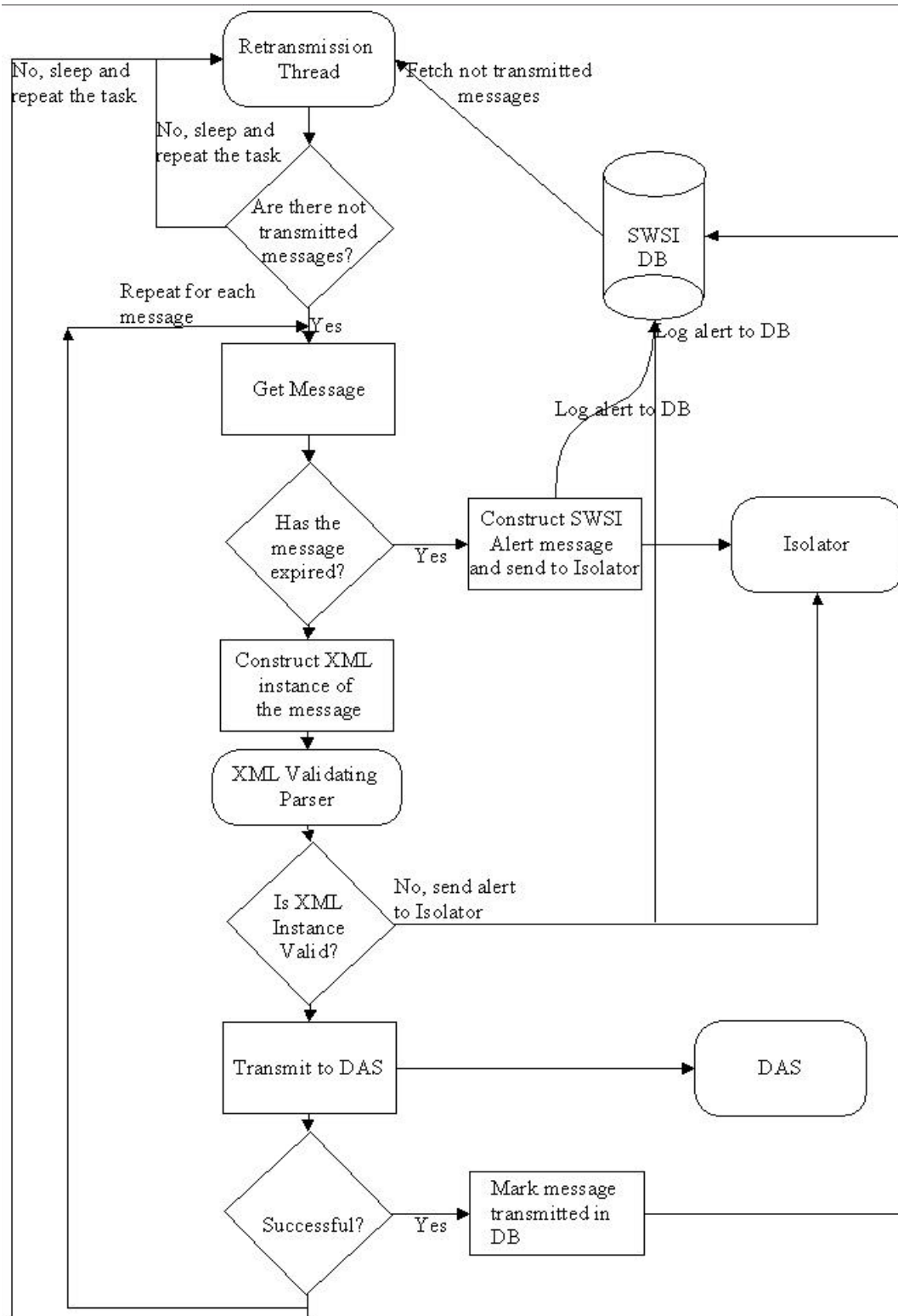


Figure 7-4 Retransmission Control Logic

7.3 Database Interface

To database access is done using standard JDBC 2.x protocol. The SDIF will be updating database tables (TBD) to set flags retransmission flags.

7.4 Support for test and operational modes

The SDIF can support test and operational modes. The SDIF achieves this capability by instantiating multiple time's subparts of itself that are responsible for communication with DASCON and Database. Those instances are different by parameters supplied to them (e.g. Host and port information). When a message from Isolator with a flag indicating test mode comes to the SDIF, then the SDIF will contact test database and messages will not be forwarded to operation DASCON, but rather just dropped or forwarded to DASCON simulator when one is available. If a message coming from Isolator does not have test flag set on, then the SDIF will operate in its normal mode contacting operation database and DASCON itself.

Section 8. Database Design

8.1 Design Principles and Guidelines

This design follows the generally accepted relational database design principles. All the data is stored in its natural Oracle type in the database; e.g. all the absolute date is stored in Oracle's "Date" type, relative times are stored in Number format and variable length strings are stored as the varchar2 type. To allow for user definable screens, some layout information is included in the database. This information includes the display order number, position, and other information to be used by the GUI screen builder. A SWSI developed Stored Procedures are used to insert into and make updates to all the dynamic data. Some of the advantages of using the Stored Procedures are that the complexity of the logical schema is hidden from the application layer, all the business rules are encapsulated in a central place, and the database integrity/consistency is assured. The use of the Stored Procedure provides enhanced performance due to the reduction of independent transactions and also gives a simplified view of the schema to the application.

8.2 The SWSI Database Design

8.2.1 Overview

Figure 8-1 through 8-3 shows the SWSI database structure in entity-relationship notation and Figure 8-4 shows the table views created to get better performance and for ease of use by the application. The relationships are implemented by using primary keys and foreign key constraints in a straightforward manner. Some tables hold static information and others are dynamically updated by SWSI application. The SWSI data administrator is responsible for the data in the static tables. Some tables are semi-static tables, meaning SWSI data administrator is responsible for initially setting them up and the SWSI application may modify some of the contents. The application software makes the calls to the Stored Procedures to modify dynamic tables. Appendix G summarizes the tables that are defined in the SWSI schema.

The information in the static tables is used for building display panels, processing NCCDS messages and storing other static data like TDRS names, SICs, SUPIDENS.

The SWSI_USER table contains information about each user of SWSI. It also contains security-related information like IP address of where the connection is made from, account expiration time, number of failed login attempts, password expiration time. Each user is assigned a group of SICs that he/she is responsible for supporting. The SIC table has all the SICs SWSI supports and the SUPIDEN table has one-to-many relationship with the SIC table. There can be multiple users supporting the same SIC.

The SSC table contains service specification code (ssc code) assigned by NCCDS for each customized configuration settings for each service type. It also contains predefined number of ssc codes to be used

with requests made for DAS. Every sscCode for each service type has multiple service parameters with a default value for each parameter. Only privileged SWSI user is allowed to modify default values for any of the DAS SSC codes.

The SSC_PARAM table contains default values of all the parameters for each service type supported by SWSI. The SWSI supports all the service types supported by NCC and DAS. The parameter values for the NCC supported service types are manually synchronized with the NCCDS/ANCC database, while they are manually entered by the SWSI data administrator for DAS services. The PROTOTYPE_EVENT_CODE table is also manually synchronized to match the prototype ids with the NCCDS/ANCC database.

The SERVICE_PARAM table contains information for generating SWSI display panels and for processing NCCDS messages. It contains for example display order, display as (i.e. text box, drop down list), default values, parameter location in the USM message. The default values for each parameter is obtained from the NCCDS/ANCC database and procedurally stored in this table. The SERVICE_TYPE table contains all the valid service types supported by SWSI. . The UPD tables contain information about processing and displaying UPD messages.

The ALERT_MESSAGE table holds all the alerts received or generated by SWSI for . The swsicomponentid field identifies the source of an alert. The SCHEDULE_CONNECTION and REALTIME_CONNECTION tables are used to establish and manage connections with NCCDS/ANCC.

All the Schedule Requests submitted by the SWSI users are kept in the REQUEST and SAR tables. The REQUEST table contains requests made by a user for requesting resources for TDRS support from NCCDS (e.g. SAR, SDR, ASAR, RR) and from DAS (e.g. RAR, RADR, RAMR, PBKR, PBKDR, PBKMR). The SR_SERVICE table is a child of SAR table and contains all the services associated with each add event scheduled. The SR_PARAMS table contains all the parameters specified in the SSC code used in the service. The parameters that are changed by the user (respecifiabiles) are flagged to facilitate construction of a SAR message. The requestId field in REQUEST is tied to Oracle's sequence counter and is incremented for each new record stored in the table.

The responses to the Schedule Requests (USMs) are stored in the ACTIVE_SCHEDULE and it's associated tables. The user Reconfiguration Requests (User GCMRs) are kept in the USR_GCMR table. The GCMR_PARAM table contains only the parameters changed by the user during a reconfiguration request. When a user makes a request for a GCMR, ACTIVE_SCHEDULE is searched to find the parameter values for the requested service (TDRS, SUPIDEN, service type & link number). The value is overridden if the parameter is also found in GCMR_PARAMS. The last accepted value found is returned.

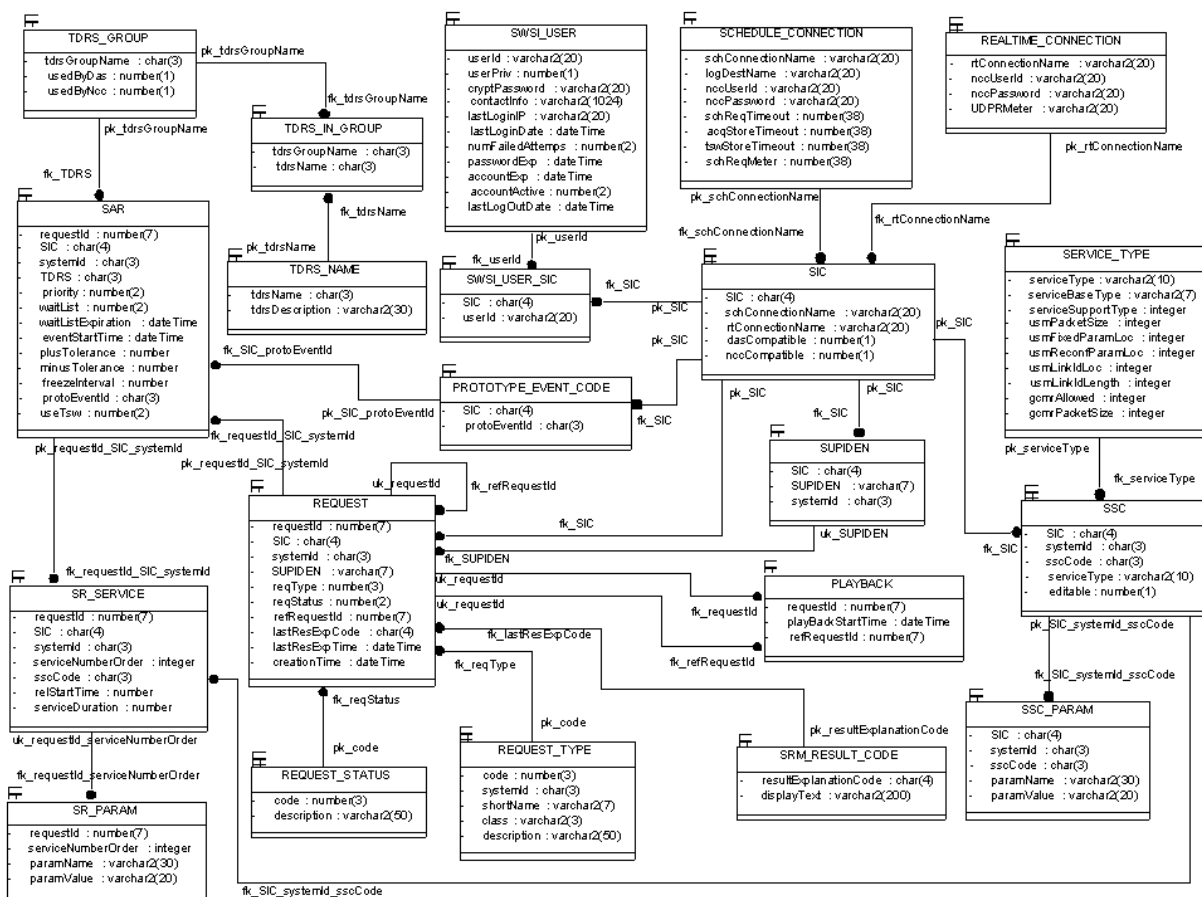


Figure 8-1 SWSI Database Schema (part 1 of 3)

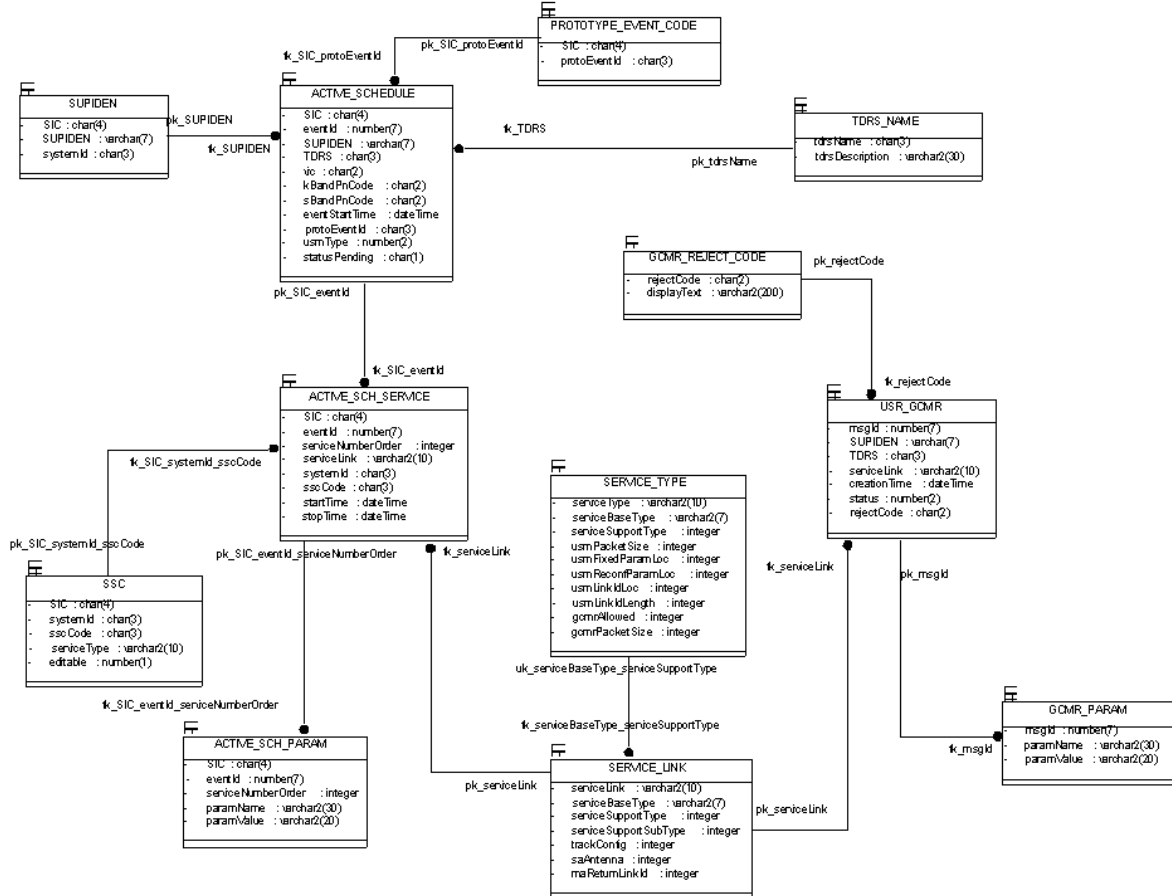


Figure 8-2 SWSI Database Schema (part 2 of 3)

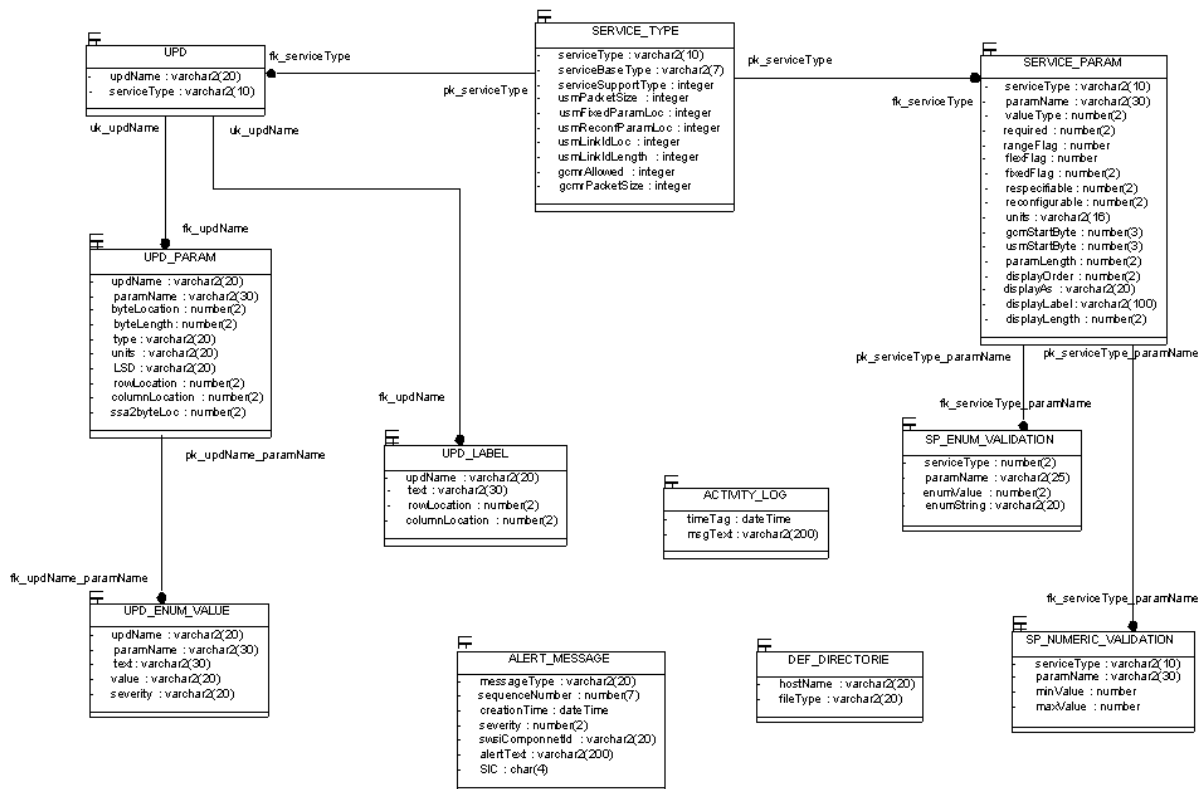


Figure 8-3 SWSI Database Schema (part 3 of 3)

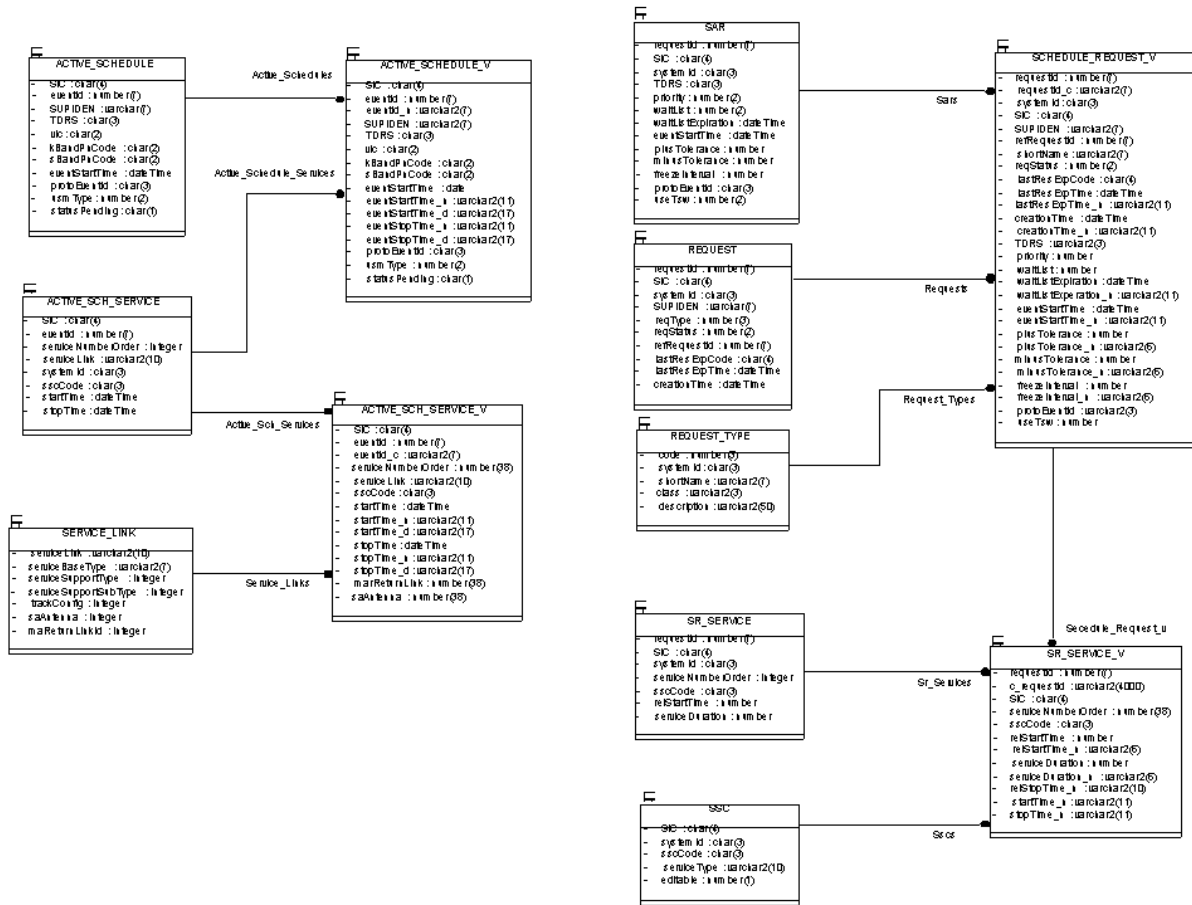


Figure 8-4 SWSI Database Table Views

8.2.2 Stored Procedures

8.2.2.1 SWSI_ActiveSchedule_pkg

Purpose: This package encapsulates a set of interface procedures for inserting active schedule records into the SWSI database. The application can use these procedures to simplify these operations and, in turn, isolate the underlying logical database representation from the application.

The following interface procedures are provided:

AddUSM - insert an active schedule record, which represents the header information of an NCC User Scheduling Message (USM).

AddService - insert an active schedule service record, which represents one of the services in the body of a USM. A USM can contain one or more services.

AddServiceParam - insert an active schedule service parameter record. An active schedule service can include zero or more "respecified" service parameters. Each service parameter is represented as a keyword:value pair, both stored in text format.

UpdateStatus - update the StatusPending field of an Active_Schedule if a Delete or Replace request was issued for a related SAR.

Purge - Purge the active schedule of events older than a specified time.

8.2.2.2 SWSI_ScheduleRequest_pkg

Purpose: This package encapsulates a set of interface procedures for inserting NCC and DAS request records into the SWSI database. The SWSI/Isolator uses these procedures to simplify these operations and, in turn, isolate the underlying logical database representation from the application.

The following interface procedures are provided:

AddSAR - insert a schedule request record, which represents the header information of an NCC User Scheduling Message (SAR) or a DAS Resource Allocation Request (RAR).

AddService - insert a schedule request service record, which represents one of the services in the body of a SAR (or RAR). A SAR can contain zero or more services. A RAR contains one service.

AddServiceParam - insert a schedule request service parameter record. A schedule request service can include zero or more "respecified" service parameters. Each service parameter is represented as a keyword:value pair, both stored in text format.

AddPlaybackRequest - insert a DAS Playback request into the Request table. The request includes one playback time and referenced SAR request ID. Additional playback times and referenced SARs can be added by invoking AddPlaybackEvent for each additional playback time period.

AddWaitListRequest - Add a pending SAR to the NCC wait list. This procedure sets the WaitListExpiration time of the related SAR. If that request is the head of a chain of SARs, then the WaitListExpiration time is replicated to all SARs in the chain.

AddDeleteRequest - Insert a schedule delete request to the SWSI database. This is an SDR for NCC requests and RADR for DAS requests.

Purge - Purge schedule and playback requests older than a specified time. As a precaution, no events are purged that are newer than yesterday.

8.2.2.3 SWSI_ScheduleResponse_pkg

Purpose: This package encapsulates a set of interface procedures for handling responses for schedule requests from either NCC or DAS. The SWSI/SNIF uses these procedures to simplify these operations and, in turn, isolate the underlying logical database representation from the application.

The following interface procedures are provided:

UpdateStatus - Update a schedule request status and/or result explanation code. If the original request was an NCC Schedule Delete Request and the response is an SRM granting the request, then this procedure also deletes related records from the Active Schedule.

8.2.2.4 SWSI_GCMR_pkg

Purpose: This package encapsulates a set of interface procedures for inserting GCMR records into the SWSI database. The application can use these procedures to simplify these operations and, in turn, isolate the underlying logical database representation from the application.

The following interface procedures are provided:

AddGCMR - insert a GCMR record, which represents the header information of an NCC Ground Control Message (GCM). The caller may furnish either (TDRS + Supiden + SSCcode + Time), or (SIC + ActiveSchedule.EventID + Service_Number)

AddGCMRparam - insert a GCMR parameter record. A GCMR can include zero or more "respecified" service parameters. Each service parameter is represented as a keyword:value pair, both stored in text format.

UpdateStatus - Update the status of a GCMR in response to a Schedule Result Message (SRM) from NCC.

8.2.2.5 SWSI_SSCEdit_pkg

Purpose: This package encapsulates an interface procedure for editing DAS SSC parameters.

The following interface procedure is provided:

EditSSCparam - Alter the default value of an existing DAS SSC parameter.

8.3 Database Configuration

The estimated minimum size for the SWSI database is 65 Mbytes for the data for ten spacecraft, plus additional space for messages, alerts, etc. Other space is also required for rollback segments and the System Global Area. At this time it is better to use the initially implemented database, either for analytical estimates or to populate it with dummy data for a number of spacecraft, rather than trying to refine the *a priori* sizing estimates.

8.4 Database Maintenance

8.4.1 Synchronization with NCCDS

The Service Specification Codes (SSCs) for the SWSI customers only, the service parameters, UPD parameters and TDRS names will be imported from the NCCDS to create the initial database. The information about the DAS SSC codes will be obtained from the DAS project and will be stored in the SWSI database by the SWSI data administrator. Subsequent synchronization will be required when any of this data is changed in the NCCDS. Procedures will be established to notify the SWSI operations of the need for an update. The method for updating the SWSI data will be determined later; it could be manual, through a PL/SQL session, rather than reloading the entire set of data from the NCCDS. Changes to this data are expected to be infrequent.

8.4.2 Purging

This database will require periodic purging of old data, specifically the schedule requests, active schedule, messages and alerts. Scripts will be developed to perform these functions. The frequency of purging will depend on the number of users, the database size, and the physical disc space available for the database.

8.4.3 Backup and Recovery

To protect the SWSI database from loss of data due to system failure, hardware failure, or media failure, a backup and recovery plan will be implemented utilizing Oracle's Enterprise Manager tool to perform periodic database backups as required. The Enterprise Manager allows for the automation of online hot backups of the database, and keeps track of the location, and the latest version of the backed

up files. By keeping track of the latest version and location of the database files, the Enterprise Manager can speed up the recovery process.

8.5 Operational Considerations

SWSI will not have any record of Schedule Add Request (SAR) initiated outside of SWSI. However, if a Schedule Delete request is made outside of SWSI on a SAR originally made from within SWSI and is accepted by the NCCDS, the corresponding request or event will be marked as deleted in the SWSI database.

Section 9. TUT Server

The TUT World Wide Web (WWW) Server provides information about unscheduled TDRS resources. It consists of start and stop times of unscheduled use of the Single Access (SA), Multiple Access Forward (MAF), and S-band Multiple Access Forward (SMAF) antennas, and Multiple Access Return (MAR) and S-band Multiple Access Return (SMAR) links for each TDRS. This data is essentially the unused time in the schedule, with a few adjustments due to flexible events with flexible start and stop times and/or flexible resources.

The customer who desires to view the TUT information uses a WWW browser to access the TUT web address. A query page is returned, on which the customer may select the time periods, services and TDRSs for the desired TUT information. When the customer selects the Submit button on the page, the query is sent to the TUT Server, which extracts the specified information based on the query, formats it on a Hypertext Markup Language (HTML) page, and returns it to the customer.

The NCCDS TUT Server provides this service only to customers located on the Closed IONET. The SWSI will extend the service to customers on the Open IONET and Internet by mirroring it to the Open SWSI Servers. The Closed SWSI Server will periodically upload the raw TUT data file and pass it through the NISN Secure Gateway in order to maintain timely TUT information on the open networks.

The Closed SWSI Server will periodically run a Java command-line web-client (TUT Proxy Client) to retrieve the raw TUT information file. This TUT Proxy Client will run as a Unix cron job on the Closed SWSI Server. The TUT Proxy Client will then initiate a connection to a stand-alone process (TUT Proxy Server) running on the Open SWSI Server. The TUT Proxy Server will listen for connections from the TUT Proxy Client on a dedicated TCP Port all the time. The TUT Proxy Server accepts the connection and stores the raw TUT data file in the proper web server directory. The updated TUT data is now available to be viewed by users on the Open IONet by the methods described previously.

Section 10. Security

10.1 Security Requirements

SWSI will adhere to the following security requirements:

- NASA Procedures and Guidelines (NPG) 2810.1, Security of Information Technology - Mission (MSN) category of NASA information, August 1999
- Security Plan for the Network Control Center, NCC 98, 451-SP-NCC/1998
- IP Operational Network (IONet) Security Plan, 290-003, September 1999

Ideally, NASA should be the digital Certificate Authority (CA). Until NASA/GSFC becomes a certificate authority, the SWSI project may have to generate its own certificates. There is a possibility that multiple certificate authorities may need to be supported.

10.2 Security Model

The SWSI Security Model will use a COTS toolkit for the following:

- Protocol:
 - Secure Socket Layer (SSL) version 3 protocol is used to provide strong security between the Client and Application Server
 - SSL is a well established, highly tested, and widely used protocol
- Authentication:
 - Strong Client authentication using signed digital certificates from a certificate authority (CA)
 - Strong Application Server authentication using signed digital certificates from a CA
- Level of Security:
 - Secure strong encrypted session key exchange
 - Ability to change session keys during a session
 - Resists “Clear Text Attacks” by using large session cipher keys (where allowed by law)
 - Defeats “Replay Attacks” by using one-time unique numbers, as part of each message, associated with each connection id.
 - Defeats “Man In The Middle Attacks” by the use of CA signed Application Server certificates
- Flexibility:
 - Security is implemented at the application level, allowing complete customization of the security model by the applications developer
 - Numerous cryptographic algorithms may be used by SSL, with minimum maintenance costs
 - Complete source code is available for the toolkit and the cryptographic algorithms

- Portability:
 - SSL toolkit is written in Java, and is portable to any platform supporting Java 1.0.2, Java 1.1, and Java 2
 - The Client and Application Server components may reside on different hardware platforms, running different operating systems
- Implementation:
 - All Clients will be required to present a verifiable certificate from a trusted CA (TBD) to the SWSI Application Server
 - The SWSI Application Server will present a verifiable certificate from a trusted CA (TBD) to each Client
 - The SWSI Client applications and SWSI Application Server will verify each other's certificates, and upon successful verification, will establish a secure SSL connection

10.3 Security Features

Security features coded into SWSI include the following (see the *Security Plan for Space Network Web Services Interface*, 452-SP-SWSI, May 10, 2000 for additional information):

1. Enforce Client password length/constraints/change frequency.
 - Minimum length 8 characters (configurable) – Isolator enforced
 - (configurable) of the following 4 criteria must exist (Isolator enforced):
 - One capital letter
 - One lower case letter
 - One numerical character
 - One non-alphanumeric character
 - Changed every 90 days (configurable) – Isolator enforced

Protocol:

When a user attempts login, the Isolator will check the password file and determine if a new password is required. The Isolator will send the Client a request for a new password, if a new password is required. The user must be able to provide his old password, and generate a new password using the criteria supplied by the Isolator, and satisfying the full criteria listed above. The Isolator will validate the user's supplied old password based on the password file. The Isolator will also quality check the new password to insure that it uses the above criteria. Either an error will be sent to the Client or the password will be accepted and stored in the password file. The password file will be updated to reflect when a new password will be required. A deactivated account will not be prompted for a new password. A new or re-activated account will always prompt for a new password. The user should have the ability on the Client side to initiate a password change. Such a request will be handled at login time, and follow the above protocol.

2. Restrict number of failed login attempts.

- The Isolator should only allow a maximum of 3 (configurable) failed login attempts, after which a user's account will be deactivated by putting a flag in the password file. An error is sent to the Client.
 - The system administrator can re-activate a user account, with a new password, using the password management utility (needs to be written). The user will need to change his password immediately upon first login.
3. Utility to manage password file.
- The password "file" may actually be a table in the database.
 - Develop a GUI based utility for password management.
 - The utility should allow users to be listed, added, deleted, deactivated, re-activated.
 - The file will maintain a userid, encrypted password, name, phone number, activation status flag, password change date, passphrase change date, and possibly other fields.
 - The utility will quality check the password for new users, as detailed in item #1.
 - A re-activated user will prompt for a new password from the system admin, which will quality check the password, as detailed in item #1.
 - The password change date will be changed to the current date for new and re-activated users, forcing users to enter new passwords upon first login.
4. Audit files.
- All logins (successful or failed) shall be logged.
 - A separate file must be maintained for the unsuccessful login attempts.
 - Client and Application Server activities will be logged for auditing purposes
 - The Application Server will have some control over the granularity of logging other data.
5. Isolator
- The SWSI Isolator will validate all requests from each SWSI Client application for adequate authority

Appendix A – Common Classes

Data will be passed between the SWSI Client, Application Server, and Isolator subsystems using serializable, common objects. This data will be passed over sockets using Object Streams. Classes used for common objects will provide a pair of get and set methods for each data item in the class. This will allow access to these data items. A class diagram for most classes is given. These objects may include instances of other objects, as shown in the class diagrams. Table A-1 presents the list of classes used to transfer data.

Class	Build	originator	notes
LoginObject	1	Client	See Figure A-1
SetupObject	1	Isolator	See Figure A-4
LogoffObject	1	Client	See Figure A-1
LoginFailed	1	Isolator	See Figure A-3
ChangePasswordRequest	1	Isolator	See Figure A-3
ChangePassPhraseRequest	1	Isolator	See Figure A-3
PasswordChanged	1	Client	See Figure A-3
PassPhraseChanged	1	Client	See Figure A-3
ChangeResponse	1	Isolator	See Figure A-3
IsoConnectionStatusChanged	1	Server	See Figure A-3
BackendConnectionStatusChanged	1	Isolator	See Figure A-3
MnemonicActivation	1	Client	Informs Application Server of requested data
MnemonicDeactivation	1	Client	Informs Application Server to stop requested data (applies only to UPD data)
MnemonicRequest	1	Client	Provides the Isolator with the information needed to fill the request. See Figure A-11.
Vector of data object(s)	1	Isolator	Objects used to transfer data
USM_List	1	Isolator	See Figure A-5
USM_SSC_List	1	Isolator	See Figure A-5
Schedule_Request_List	1	Isolator	See Figure A-5
SAR	1	Client	See Figure A-6
SDR	1	Client	See Figure A-7
Alert	1	Isolator	See Figure A-2

Table A-1. List of Common Classes

Class	Build	originator	notes
GCParms	2	Isolator	See Figure A-5
TSW	2	Client	See Figure A-1
SV	2	Client	See Figure A-1
RAR, RADR, RAMR, PBKR, PBKDR, PBKMR	2	Client	DAS Resource and Playback Requests. See Figure A-12
RAV, PBKS	2,3	Isolator	DAS Resource and Playback Availabilities. See Figure A-13
ModifySSC, UnlockSSC	2	Client	SSC Requests. See Figure A-14
ViewSSC	2	Isolator	SSC Values. See Figure A-14
SigRR, URRM	2	Client	DAS GCMRs. See Figure A-15
Service_Reconfiguration_Request	2	Client	See Figure A-2
User_Reacquisition_Request	2	Client	See Figure A-2
Forward_Link_Sweep_Request	2	Client	See Figure A-2
Forward_Link_EIRP_Reconfiguration	2	Client	See Figure A-2
Doppler_Compensation_Inhibit_Request	2	Client	See Figure A-2
Expanded_User_Frequency_Uncertainty_Request	2	Client	See Figure A-2
UPD	2	Isolator	See Figure A-2
ASAR	2	Client	See Figure A-8
RR	2	Client	See Figure A-9
WLR	2	Client	See Figure A-10
TTM	2	Isolator	See Figure A-2
RCTD	2	Isolator	See Figure A-2

Table A-1. List of Common Classes (cont.)

Except for an initial setup object and occasional password changing support, all data sent to the Client application from the Isolator will be requested by the Client using a standard naming convention. The proposed naming convention is given in the table below. ‘EIF’ indicates the test database, ‘norm’ the standard database.

Name	Description
USM_List_(EIF or norm)_(User ID)	The active schedule list for a particular user
USM_SSC_List_(EIF or norm)_(Event ID)	The service list for a particular event
Schedule_Request_List_(EIF or norm)_(User ID)	The schedule request list for a particular user
RAV_(User ID)_####	The DAS Resource Availabilities
PBKS_(User ID)_####	The DAS Playback Availabilities
SSC_(User ID)_####	Values for a particular SSC set

GCParms_(EIF or norm)_(TDRS ID)_(supiden)_(Service Type)_(link)	The current ground control service parameter values for an event and service defined by the TDRS ID, supiden, service type, and link number.
UPD_(EIF or norm)_(SIC)	The current UPDs for that SIC
Alert_(EIF or norm)_(SIC)	Alerts for a particular SIC
RCTD_(EIF or norm)_(SIC)	The RCTDs for that SIC
TTM_(EIF or norm)_(SIC)	The TTMs for that SIC

Table A-2. Standard Naming Convention

All the common objects used to send data to the Client will implement the DataEncapsulation interface. This provides methods to set and get the name of the data. The common objects will provide enumeration constants where applicable.

All common objects used to send data to the Isolator, such as the SAR, GCMRs, SVs, and TSWs will implement the UserDirective interface. This provides methods to set and get the userID (the ID of the user sending the data). This ID will be set by the Application Server before forwarding the data on to the Isolator.

The contents of most of the above types of common objects are presented in Figures A-1 through A-10.

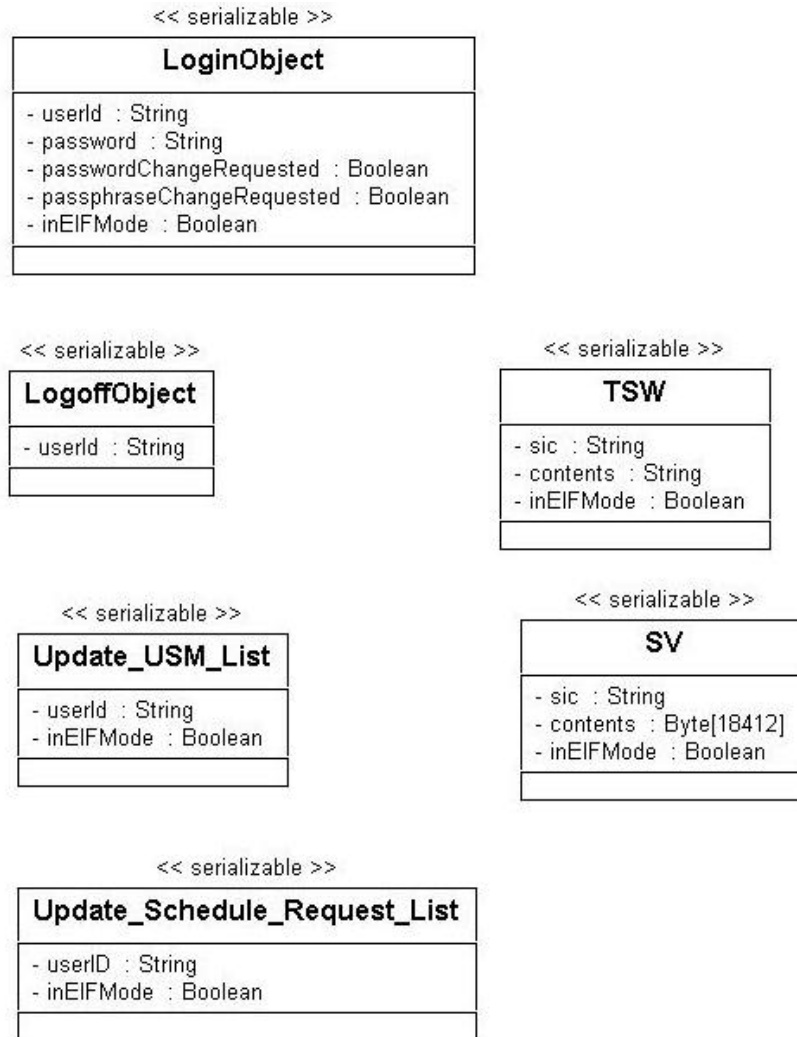


Figure A-1 Common Class Diagram 1

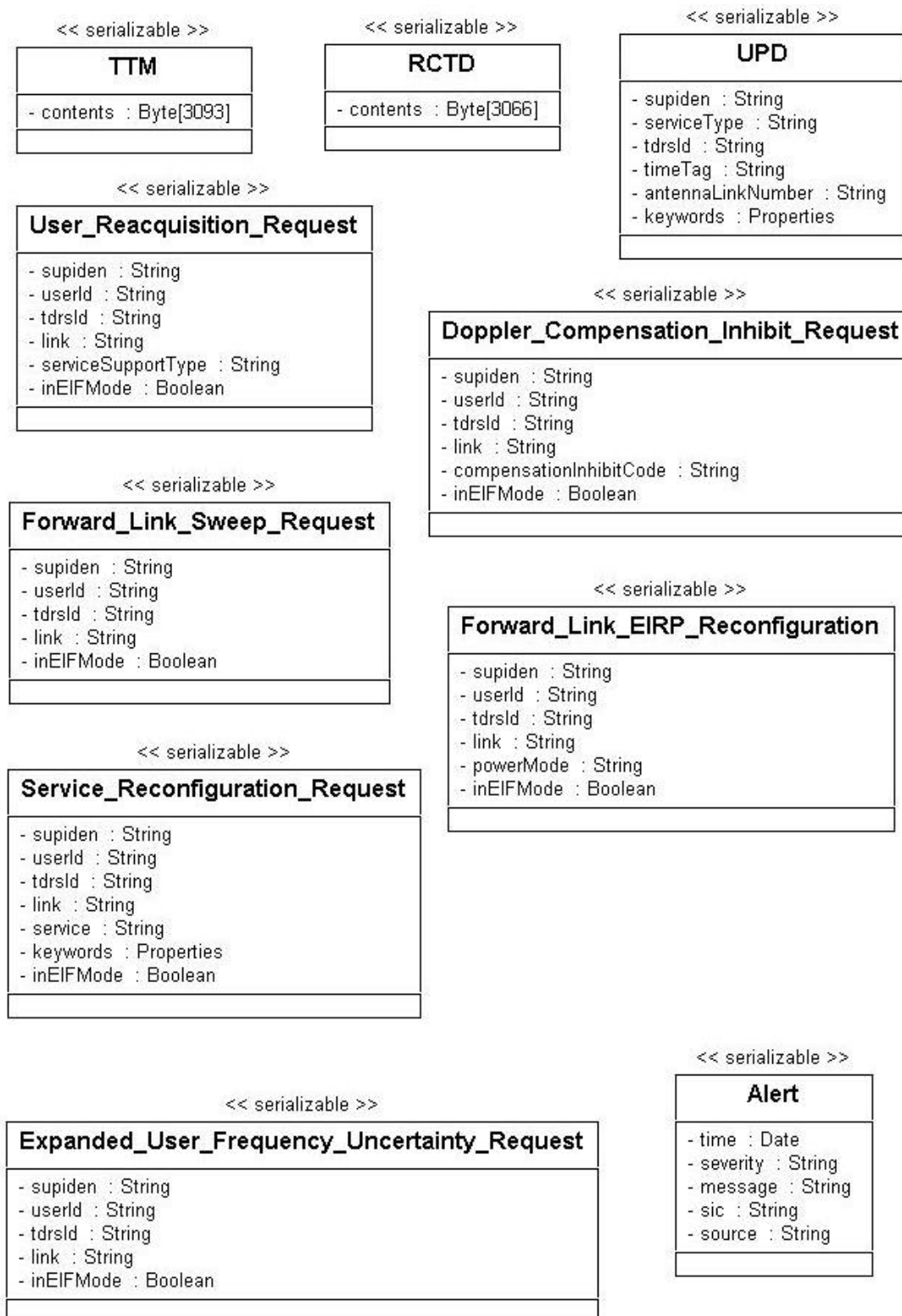


Figure A-2 Common Class Diagram 2

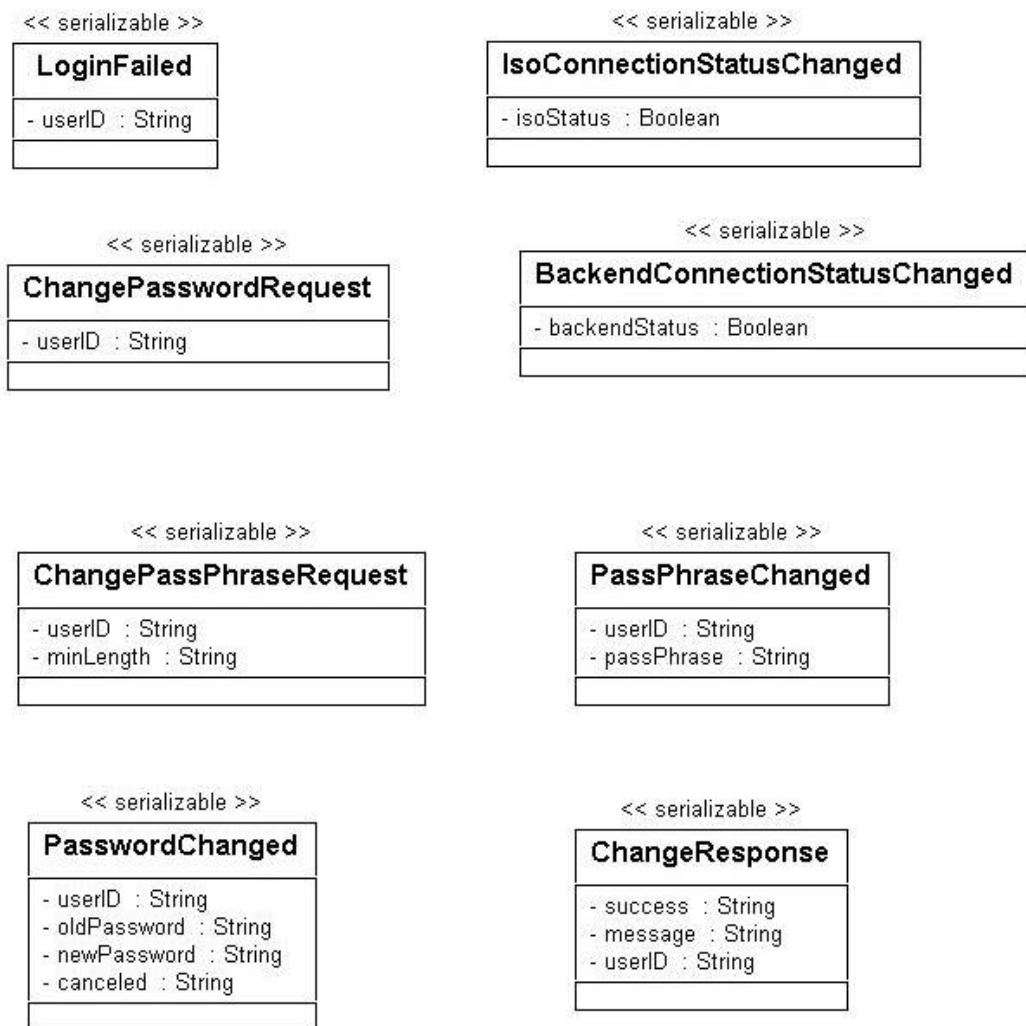


Figure A-3 Common Class Diagram 3

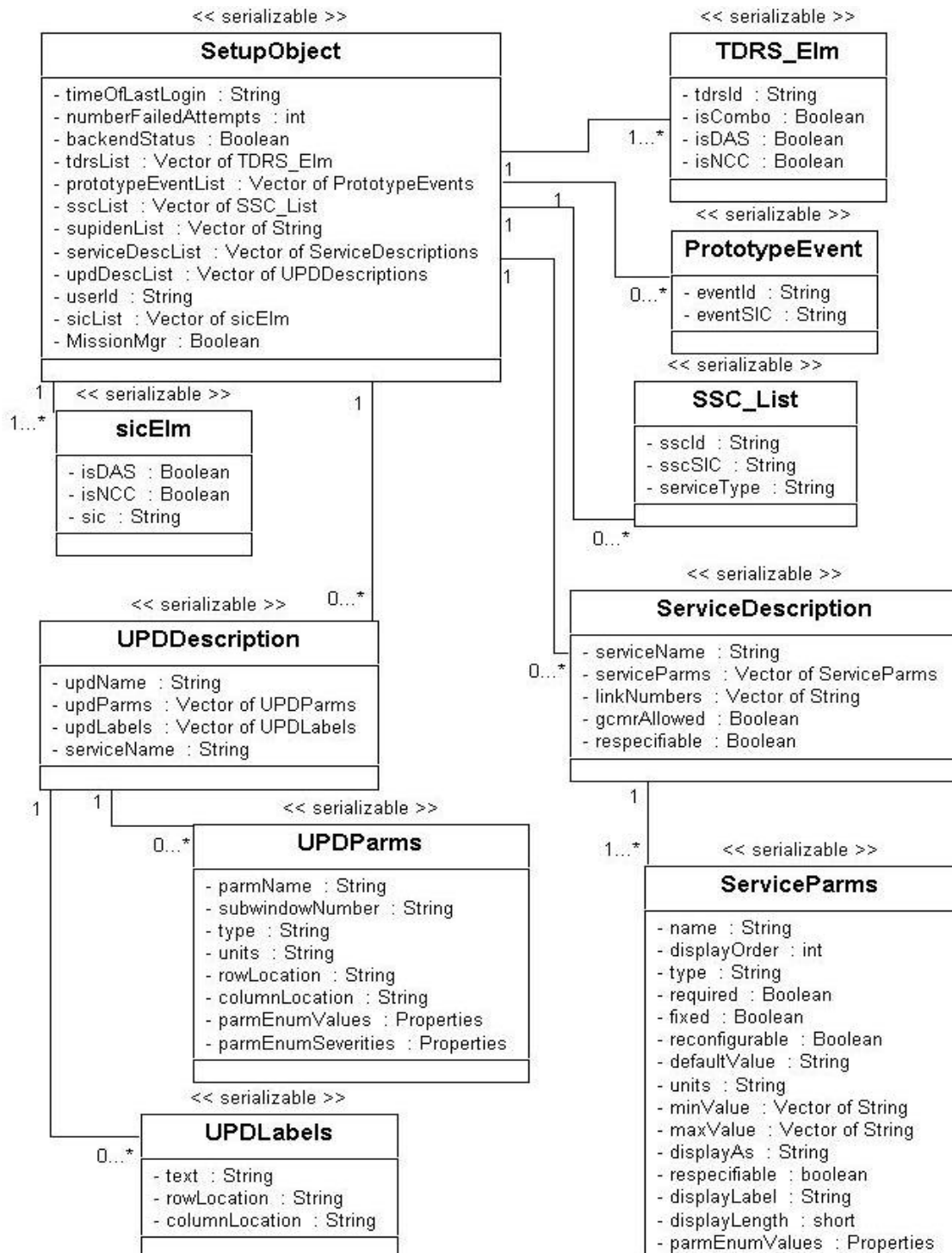


Figure A-4 Common Class Diagram 4

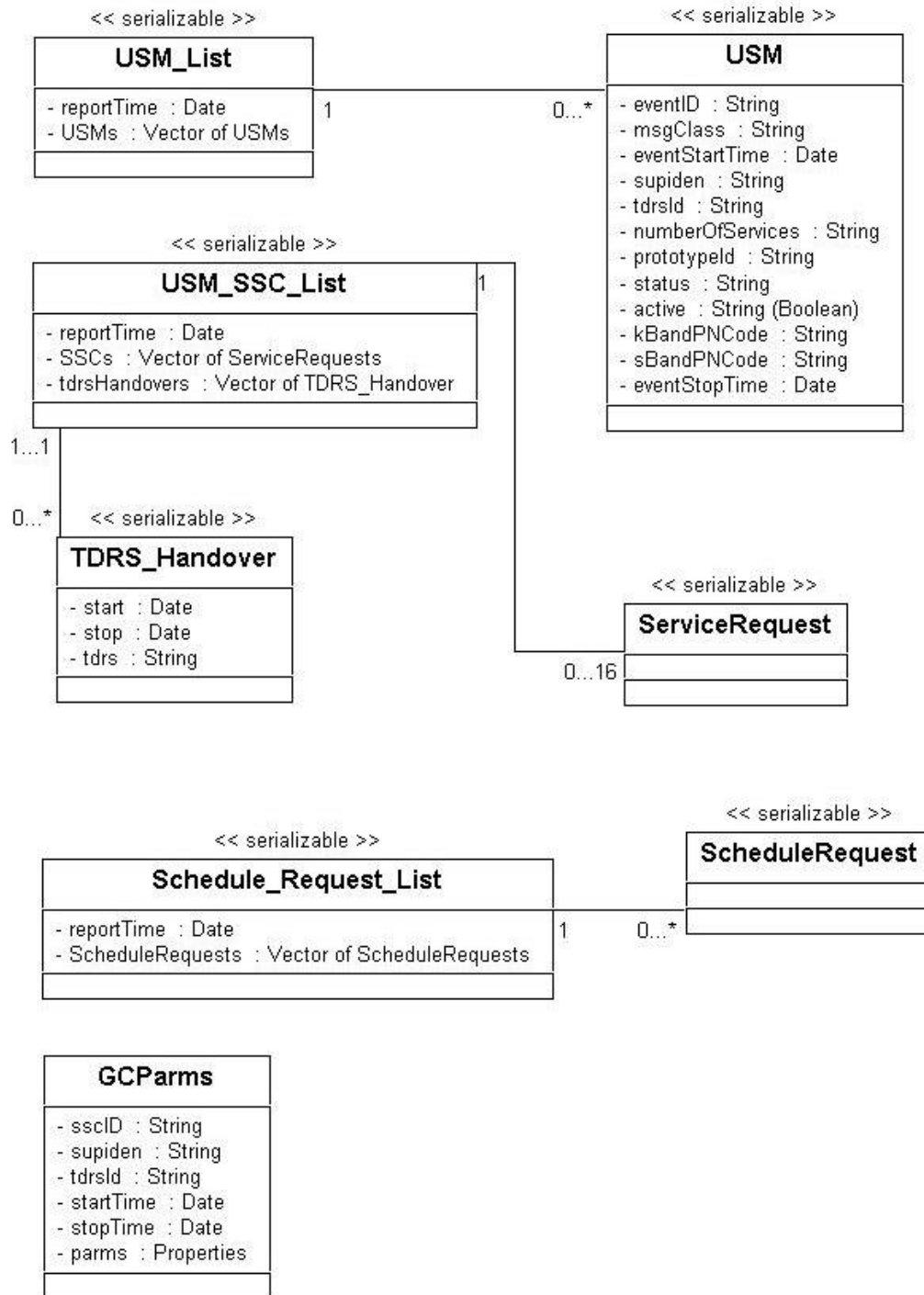


Figure A-5 Common Class Diagram 5

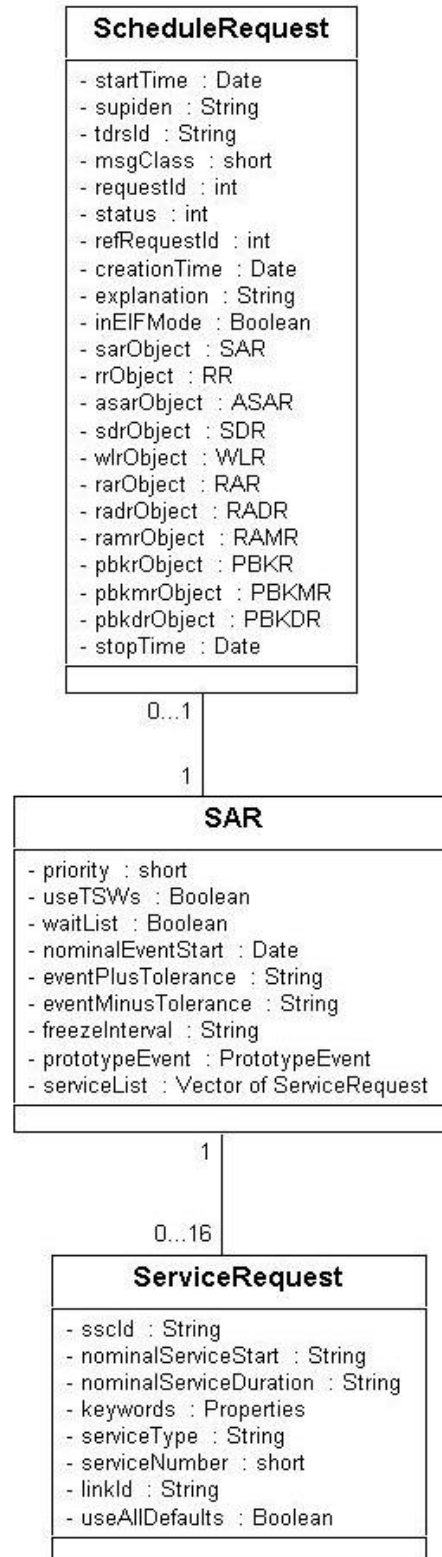


Figure A-6 SAR Common Class Diagram

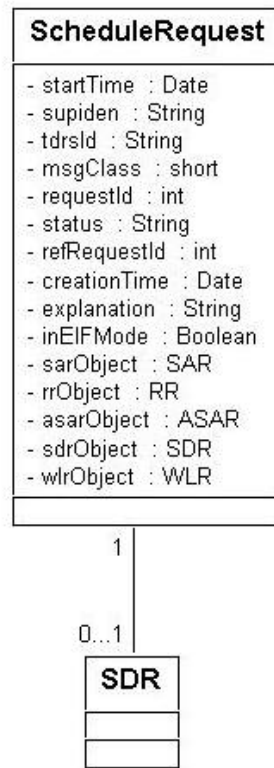


Figure A-7 SDR Common Class Diagram

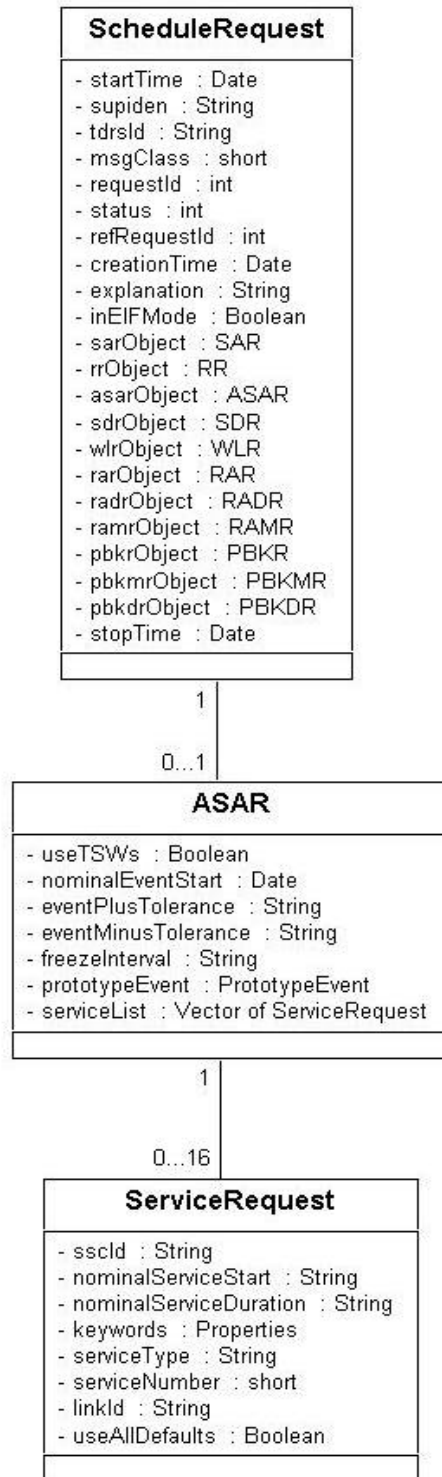


Figure A-8 ASAR Common Class Diagram

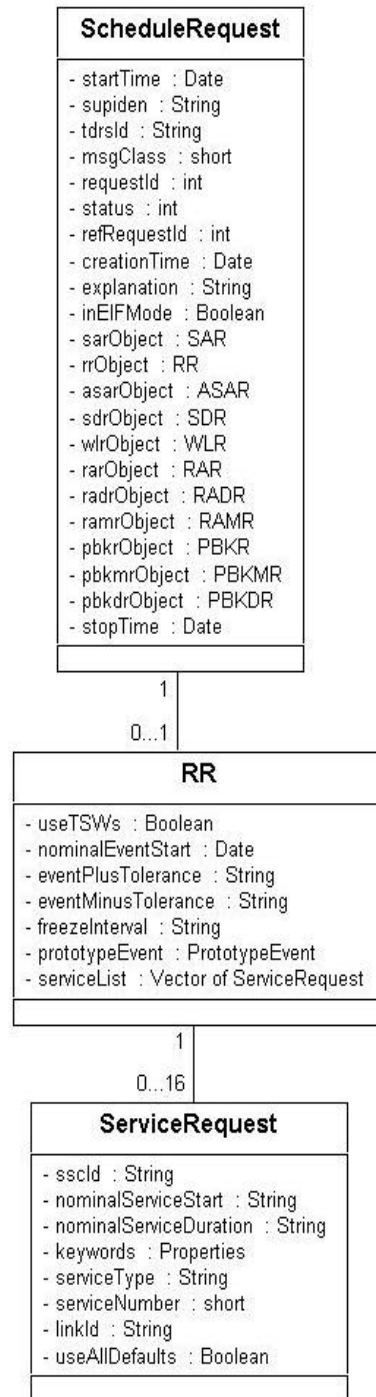


Figure A-9 RR Common Class Diagram

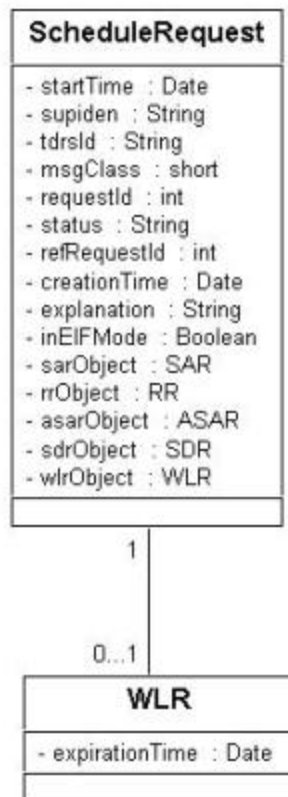


Figure A-10 WLR Common Class Diagram

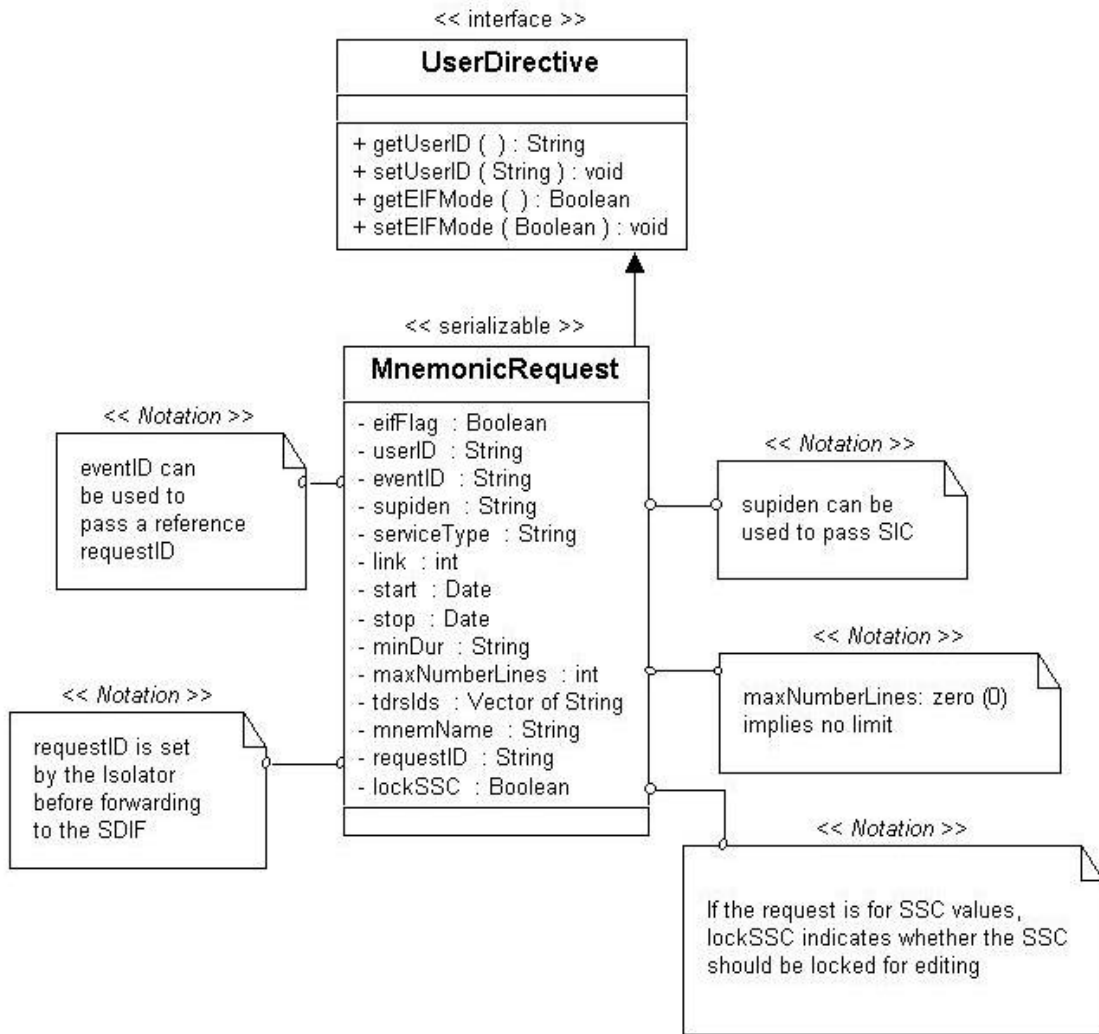


Figure A-11 MnemonicRequest Common Class Diagram

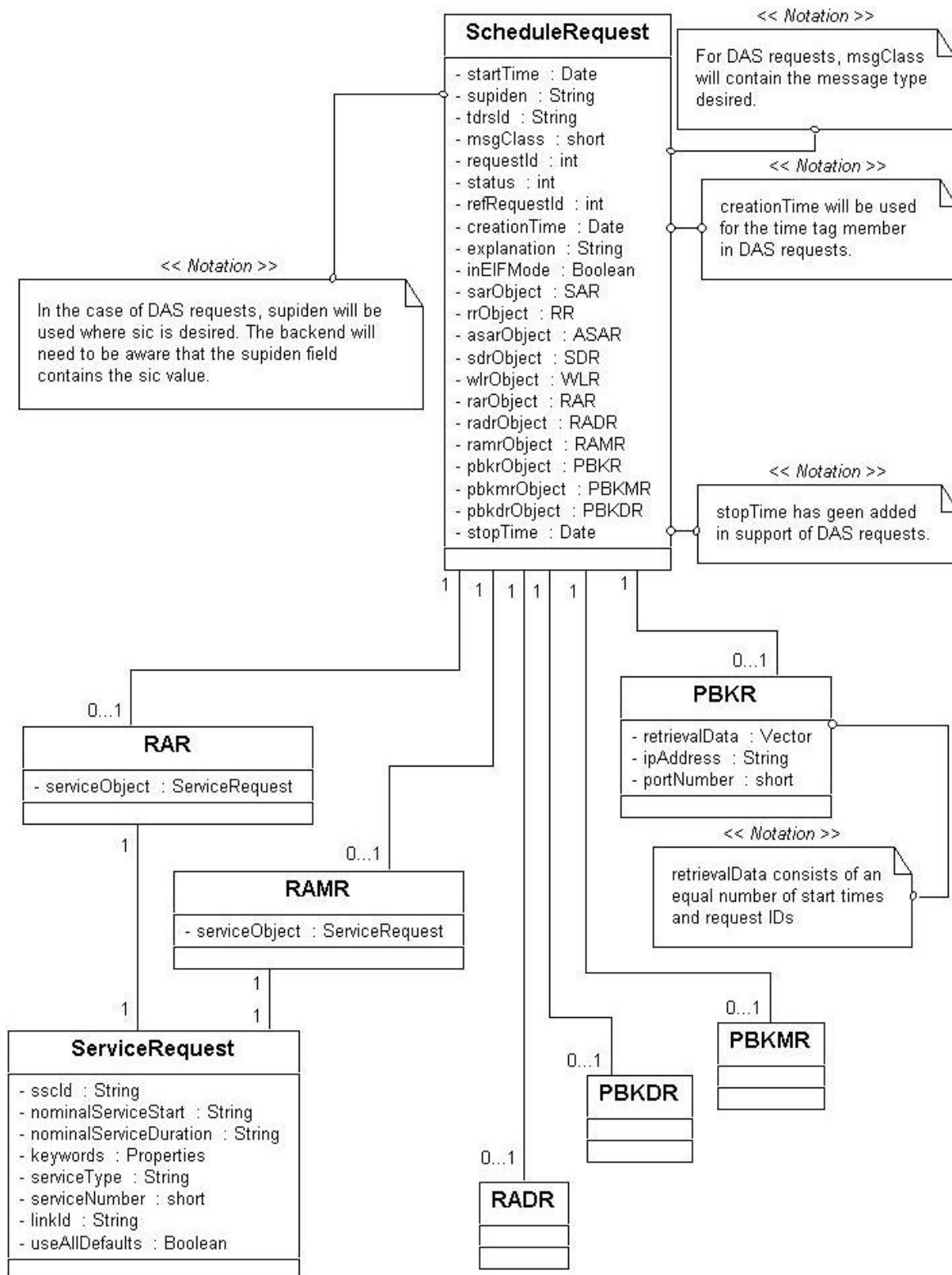


Figure A-12 DAS Requests Common Class Diagram

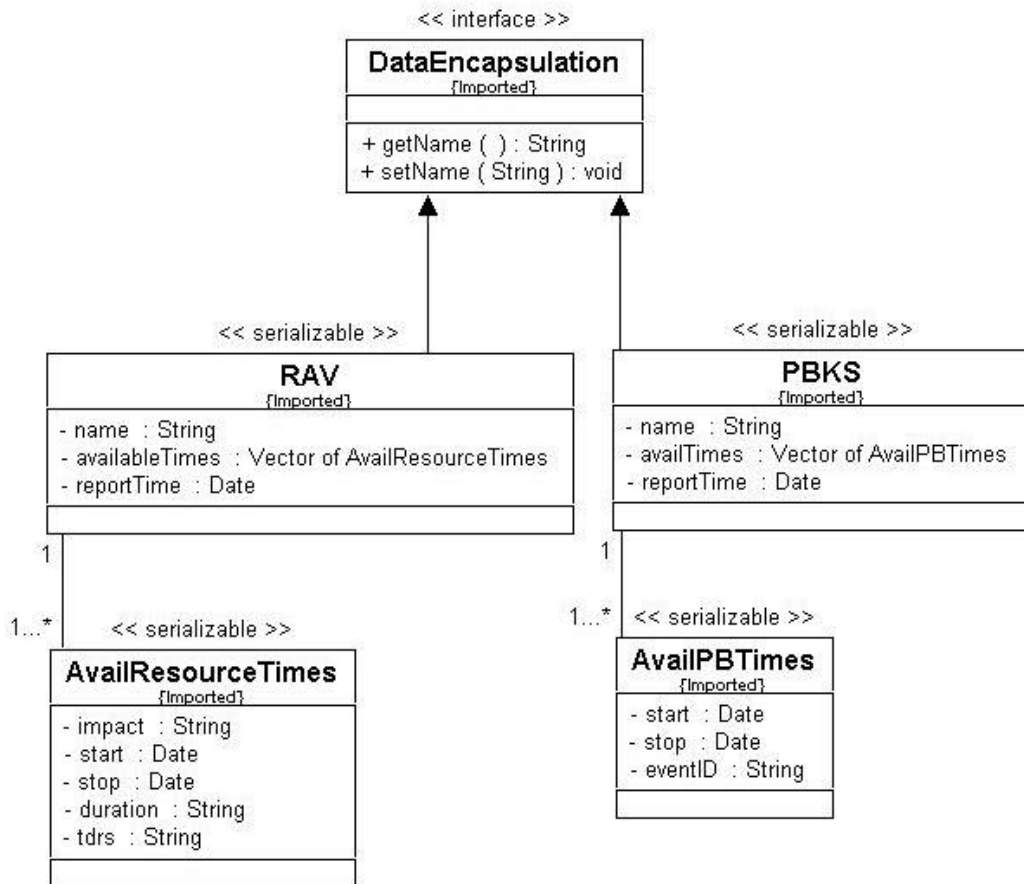


Figure A-13 DAS Availability Common Class Diagrams

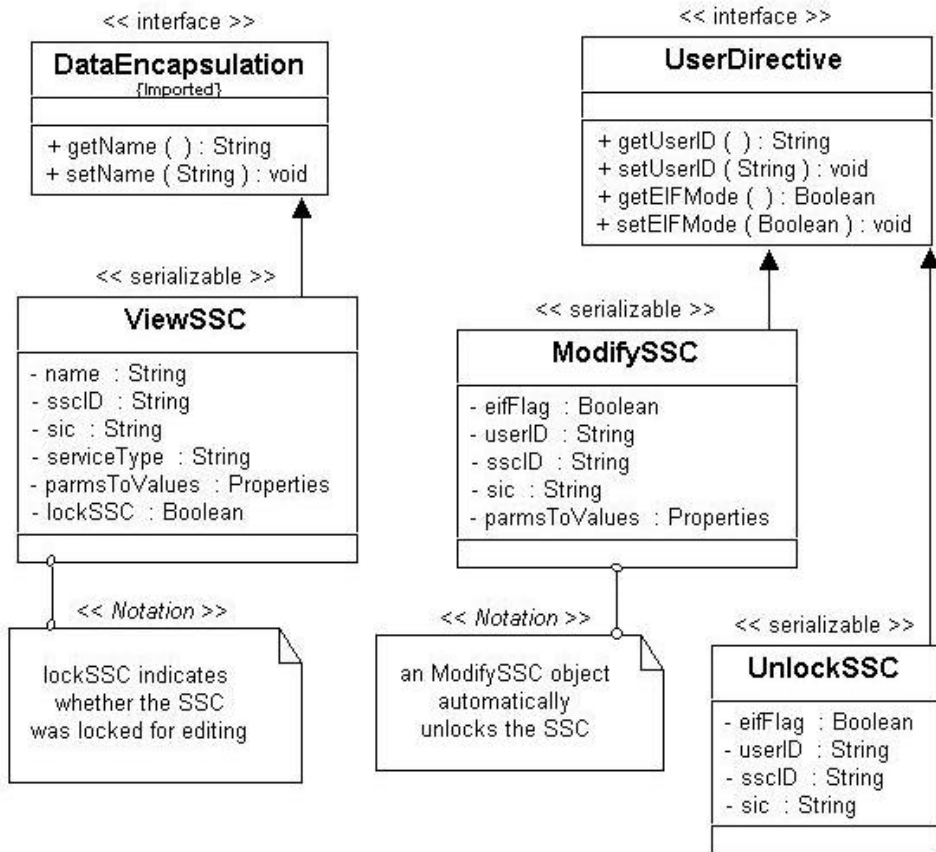


Figure A-14 SSC Support Common Class Diagrams

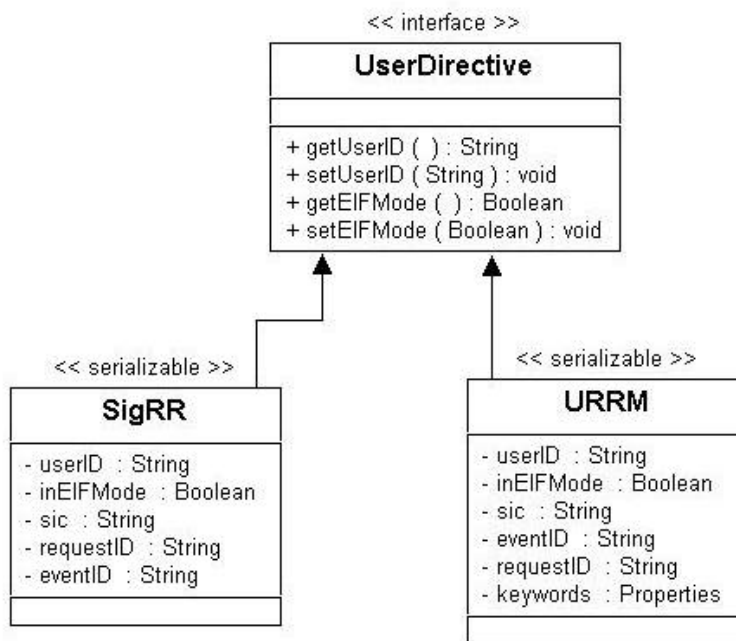


Figure A-15 DAS GCMR Support Common Class Diagrams

Appendix B - Traceability

Appendix C - Isolator-SNIF Interface

This appendix defines data exchange formats between the Isolator and the SNIF.

The communication protocol between the Isolator and the SNIF is UDP, the format of the data exchanged between the two elements is chosen to minimize software changes in the future in case we decide to change the protocol to TCP/IP.

In general, the format will consist of 2-bytes Synchronization pattern, followed by 4 bytes of support identification code, followed by 1-byte of NCC mode, followed by 1-byte of data type, followed by 2-bytes of data length and finally followed by variable length of the actual data information.

SY	sic	n	t	dl	Data Information
----	-----	---	---	----	------------------

SY – 2-bytes fixed synchronization pattern. We select 2 ASCII characters ‘SY’ for sync.

sic – 4-bytes (4 Alpha-Numerical Characters) representing the Support Identification Code (SIC) specific to the mission. When the message is not mission specific (i.e. general alert for all), the SIC will be set to 0000.

n – 1 byte representing the addressed NCC that the Data Information pertain to.

- ‘N’ for normal NCC
- ‘E’ for engineering interface (EIF) or ANCC

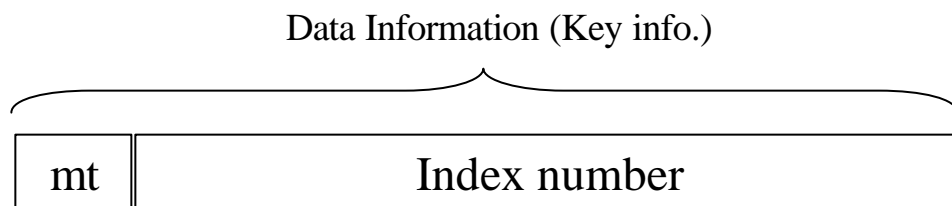
t – 1 byte representing the Type of the Information and described as follow:

- ‘K’ for Key Information of the messages stored in the data base
- ‘F’ for File Information of the messages stored in the local disk area
- ‘A’ for Alerts messages stored in the data base
- ‘M’ for actual NCC Messages such as: MGCMRs

dl – 2-bytes representing the length (in bytes) of the Data Information. The two bytes represent a binary unsigned short (16 bits) Integer value.

Data Information – variable number of bytes corresponding to the actual data exchanged between the Isolator and the SNIF. This Data Information is further defined according to its specific type as follow:

Key Information ‘K’ Type:



mt – 2-bytes (2 ASCII characters) representing the Message Type, defined as follow:

- ‘SA’ - Schedule Add Request (SAR) Messages
- ‘AS’ - Alternate SAR (ASAR) Messages
- ‘SD’ - Schedule Delete Request (SDR) Messages
- ‘RR’ - Replace Request (RR) Messages
- ‘WL’ - Wait List Request (WLR) Messages
- ‘UR’ - User Reconfiguration Request (URRM) Messages

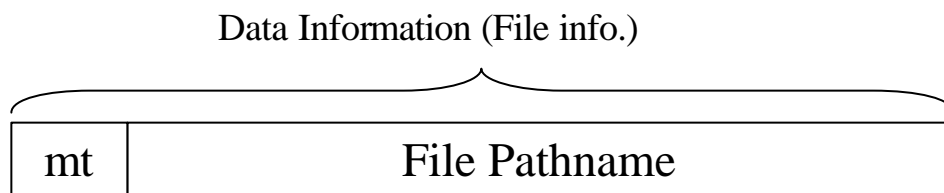
Sent by:
Isolator (SnifInterface)

Index number – Long integer (4 bytes) representing the primary key number of the message as stored in the database. The Index number is unique for each message stored in the database table. The Request ID (7-digits numerical number) will be used as the unique index number of the message. However, the valid Request ID range has to be between 1 and 8,999,999. A sequence number generated by the database with maximum and minimum limitations will be used by the Isolator, the SNIF and the SDIF as a requestID. Because some events can be indefinite (in the case of DAS), the database will check if a Request ID is still occupied by an event and, in that case, it will skip that ID and tries the next one till it finds an unused one.

The K type messages are only sent by the Isolator to the SNIF. Even though, SNIF stores messages data in the data base, it does not send K type information to the Isolator but instead it sends alerts messages. SNIF will store its own generated alerts messages in the database prior to their transmission to the Isolator.

Both the SNIF and the Isolator will poll the database or file directory periodically (every 30 - 60 seconds) for messages that have been saved but not transmitted. The messages will be processed and transmitted to the expecting end. The messages that have been stored in the data base will be tagged as done and the messages that have been stored in the local disk space will be deleted after transmission.

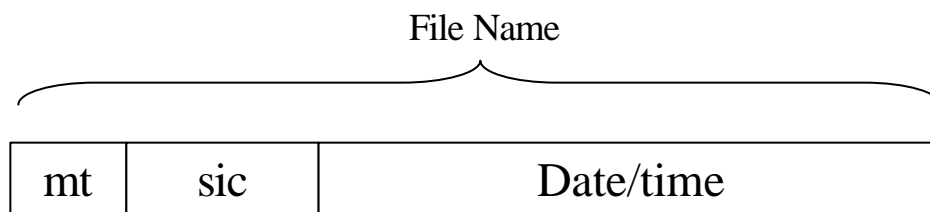
File Information ‘F’ Type:



mt – 2-bytes (2 ASCII characters) representing the Message Type, defined as follow:

- ‘SV’ - State Vector (SV) Messages
 - ‘TS’ - TDRS Scheduling Window (TSW) Messages
 - ‘RC’ - Return Channel Time Delay Measurement (RCTDM) Messages
 - ‘TT’ - Time Transfer (TTM) Messages
- } Sent by:
Isolator (SnifInterface)
- } Sent by:
SNIF

File Pathname – Variable number of bytes (ASCII characters) representing the path name of the file in the local disk area where the message was stored. The name of the file is formatted such that the SIC, Message type and date/time of storage are all included in a 15 characters string (see figure below.)



Where:

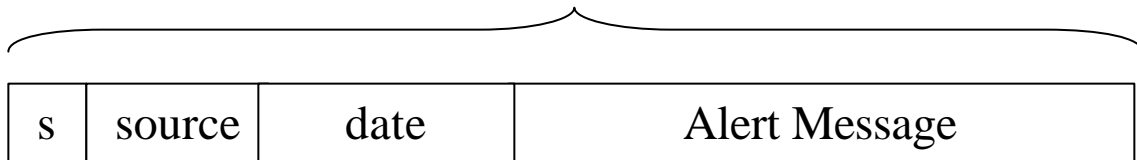
‘mt’ field is as defined above.

‘sic’ – 4-bytes representing the Support Identification Code (SIC) specific to the mission. This field is pulled from the primary header of the message just after the SY field (see format diagram on first page of this section)

Date/Time as: YYYYDDDDHHMMSS (Year, day of the year, hour, minute, second)

Alert Message ‘A’ Type:

Data Information (Alert Message)



s – 1-byte (1 ASCII character) representing the severity of the alert message, defined as follow:

- ‘G’ – Green
- ‘Y’ – Yellow
- ‘R’ – Red

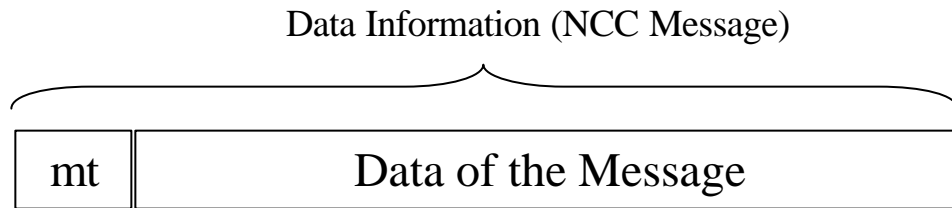
source – 16-byte string (padded with spaces) indicating the source of the alert

date – 9-byte ASCII digits representing date and time when the alert was generated in the following format: YYYYDDHMMSS (Year, day of the year, hour, minute, second)

Alert Message – Character string describing the alert message. The Alert Message can be of any length and can have any number of delimited characters.

- ‘US’ - User Schedule (USM) Messages
 - o ‘U1’ – Normal-Fixed (US1) Messages
 - o ‘U2’ – Premium-Fixed (US2) Messages
 - o ‘U3’ – Simulation-Fixed (US3) Messages
 - o ‘U4’ – Normal-Flexible (US4) Messages
 - o ‘U5’ – Simulation-Flexible (US5) Messages
 - ‘GD’ - GCM Disposition (GCMD) Messages
 - ‘GS’ - GCM Status (GCMS) Messages
 - ‘AF’ - Acquisition Failure Notification (AFN) Messages
 - ‘SR’ - Schedule Result (SRM) Messages
 - Other type of alert messages sent by the Isolator to SNIF or to the Application Server such as: event message 5 minutes before operation start time.
- Sent by:
SNIF

NCC Message ‘M’ Type:



mt – 2-bytes (2 ASCII characters) representing the Message Type, defined as follow:

- Multiple Ground Control Message Request (MGCMR) Messages
 - o ‘AC’ - User Reacquisition Request (URR) Messages
 - o ‘LS’ - Forward Link Sweep Request (FLSR) Messages
 - o ‘LE’ - Forward Link EIRP Reconfiguration (FLER) Messages
 - o ‘UF’ - Expanded User Frequency Uncertainty Request (EUFUR)
 - o ‘DC’ - Doppler Compensation Inhibit Request (DCIR) Messages
 - ‘PD’ - User Performance Data (UPD) Messages
- } Sent by:
Isolator
- } SNIF

Data of the message – variable length buffer containing the actual Message data. The data will be composed of series of all ASCII name-value pairs delimited by a semicolon ‘;’ character. The name and the value can be of any meaningful ASCII string separated by the ‘=’ sign. i.e.

MESSAGE_TYPE=91;MESSAGE_CLASS=01;MESSAGE_ID=ABCDEFGH;

Appendix D – Isolator-SDIF Interface

TBS

Appendix E – Isolator Object Types Description

Data flow between the Application Server, Isolator and SNIF

Message Name	Object Name	Application Server	Isolator	SNIF
Alert Messages	Alert	Receive Port 3	Store, Send and Forward	Store & Send
Schedule Add Request	ScheduleRequest of SAR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Schedule Delete Request	ScheduleRequest of SDR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Alternate SAR	ScheduleRequest of ASAR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Replace Request	ScheduleRequest of RR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Wait List Request	ScheduleRequest of WLR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
State Vector	SV	Send Port 1	Store data in File Send File Info	Receive File Info
TDRS Scheduling Window	TSW	Send Port 1	Store data in File Send File Info	Receive File Info
Service Reconfiguration	Service_Reconfigur ation_Request	Send Port 1	Store data and Send Key Info	Receive Key Info
GCMR- User Reaction Request	User_Reacquisition _Request	Send Port 1	Forward	Receive Key Info
GCMR- Forward Link Sweep	Forward_Link_Req uest	Send Port 1	Forward	Receive Key Info
GCMR-Forward Link EIRP Reconfiguration Normal Request	Forward_Link_EIR P_Reconfiguration (Normal power mode set)	Send Port 1	Forward	Receive Key Info
GCMR-Forward Link EIRP Reconfiguration High Power	Forward_Link_EIR P_Reconfiguration (High power mode set)	Send Port 1	Forward	Receive Key Info

Message Name	Object Name	Application Server	Isolator	SNIF
GCMR- Expanded User Frequency Uncertainty	Expanded_User_Fr equency_Uncertaint y_Request	Send Port 1	Forward	Receive Key Info
GCMR-Doppler Compensation Inhibit Request none SSA shuttle	Store data and Send Key Info	Send Port 1	Forward	Receive Key Info
Return Channel Time Delay Message	MnemonicRequest RCTDM	Receive Port 2	Receive File Info Convert file data Send Object	Store data in File Send File Info
Time Transfer Message	MnemonicRequest TTM	Receive Port 2	Receive File Info Convert file data Send Object	Store data in File Send File Info
User Performance Data	MnemonicData of UPD	Receive Port 2	Receive data Convert to Object Send Object	Send data
Schedule Result Message	SRM	Receive Alert Port 3	Forward Alert	Update Database, Store & send an Alert
User Schedule Message	USM	Receive Alert Port 3	Forward Alert	Update Database, Store & send an Alert
Acquisition Failure Notification	AFN	Receive Alert Port 3	Forward Alert	Store & send an Alert

Data flow between the Application Server and Isolator

Message Name	Object Name	Application Server	Isolator	SDIF
Schedule Request Summary Request	MnemonicRequest (Schedule_Request_List)	Send Port 1	Receive and process Request	-
Schedule Request Summary Response	MnemonicData of Schedule_Request_List	Receive Port 2	Send Object	-
Active Schedule Summary Request	MnemonicRequest (USM_List)	Send Port 1	Receive and process Request	-
Active Schedule Summary Response	MnemonicData of USM_List	Receive Port 2	Merge Data from DAS if DAS User Send Object	-
Service List for Events Request	MnemonicRequest (USM_SSC_List)	Send Port 1	Receive and process Request	-
Service List of Events Response	MnemonicData of USM_SSC_List	Receive Port 2	Send Object	-
User Log-in Request	LoginObject	Send Port 1	Receive and process Request	-

User Log-out Request	LogoffObject	Send Port 1	Receive and process Request	-
-------------------------	--------------	----------------	--------------------------------	---

Message Name	Object Name	Application Server	Isolator	SDIF
Fail Log-in Response	LoginFailed	Receive Port 2	Send Object	-
Good Log-in Response	SetupObject	Receive Port 2	Send Object	-
SSC Service Parameters Request	MnemonicRequest (SSC)	Send Port 1	Receive and process Request	-
SSC Service Parameters Reply	MnemonicData of ViewSSC	Receive Port 2	Send Object	-
Modify SSC Service Parameters Request	ModifySSC	Send Port 1	Receive and process Request	-
Unlock the SSC code	UnlockSSC	Send Port 1	Receive and process Request	-
Ground Control Message Param Request	MnemonicRequest (GCParms)	Send Port 1	Receive and process Request	-
Ground Control Message Param Response	MnemonicData of GCParms	Receive Port 2	Send Object	-

Data flow between the Application Server, Isolator and SDIF

Message Name	Object Name	Application Server	Isolator	SDIF
Alert Messages	Alert	Receive Port 3	Store, Send and Forward	Store & Send
Resource Availability Request	MnemonicRequest (RAV)	Send Port 1	Forward	Receive
Resource Availability Response	MnemonicData of RAV	Receive Port 2	Forward	Send
Active Schedule Summary Request	MnemonicRequest (USM_List)	Send Port 1	Receive, send Request and process Request	Receive & Send Active Schedule List
Active Schedule Summary Response	MnemonicData of USM_List	Receive Port 2	Send Object	-
Service List for Events Request	MnemonicRequest (USM_SSC_List)	Send Port 1	Forward	-
Service List of Events Response	MnemonicData of USM_SSC_List	Receive Port 2	Send Object	-
Resource Allocation Request	ScheduleRequest of RAR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Resource Allocation Response	RARRes	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
Resource Allocation Deletion Request	ScheduleRequest of RADR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Resource Allocation Deletion Response	RADRes	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
Resource Allocation Modification Request	ScheduleRequest of RAMR	Send Port 1	Store data in DB Send Key Info	Receive Key Info

Message Name	Object Name	Application Server	Isolator	SDIF
Resource Allocation Modification Response	RAMRes	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
Service Reconfiguration Request	Service_Reconfiguration_Request	Send Port 1	Forward	Receive
Service Reconfiguration Response	SerRRes converted to Alert	Receive Alert Port 3	Forward Alert	Store & Send an Alert
Signal Reacquisition Request	User_Reacquisition_Request	Send Port 1	Forward	Receive
Signal Reacquisition Response	SigRRes converted to Alert	Receive Alert Port 3	Forward Alert	Store & Send an Alert
User Performance Data Status	MnemonicData of UPDS	Receive Port 2	Forward	Send
Playback Search Request	MnemonicRequest (PBKS)	Send Port 1	Forward	Receive
Playback Search Response	MnemonicData of PBKS	Receive Port 2	Forward	Send
Playback Request	ScheduleRequest of PBKR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Playback Response	PBKRes converted to Alert	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
Playback Deletion Request	ScheduleRequest of PBKDR	Send Port 1	Store data in DB Send Key Info	Receive Key Info
Playback Deletion Response	PBKDRes converted to Alert	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
Playback Modification Request	ScheduleRequest of PBKMR	Send Port 1	Store data in DB Send Key Info	Receive Key Info

Playback Modification Response	PBKMRes converted to Alert	Receive Alert Port 3	Forward Alert	Update DB, Store & Send an Alert
State Vector Update	SV	Send Port 1	Forward	Receive
State Vector Update Response	SVRes	Receive Alert Port 3	Forward Alert	Store & Send an Alert

Appendix F – SWSI Database Tables

Table Name	Type of Table	Description
ACTIVE_SCHEDULE	Dynamic	Stores information from NCCDS USMs
ACTIVE_SCH_PARAM	Dynamic	Stores information from NCCDS USMs
ACTIVE_SCH_SERVICE	Dynamic	Stores information from NCCDS USMs
ACTIVITY_LOG	Dynamic	Stores User log in activities
ALERT_MESSAGE	Dynamic	Stores all the Alerts received from DAS and generated by SWSI
DEF_DIRECTORIE	Static	
GCMR_PARAM	Dynamic	Stores information about NCCDS User GCMR
GCMR_REJECT_CODE	Static	
PLAYBACK	Dynamic	Stores DAS Playback Requests
PROTOTYPE_EVENT_CODE	Static	Stores NCCDS Prototype Codes
REALTIME_CONNECTION	Static	Stores information about making connections with NCCDS real-time system
REQUEST	Dynamic	Stores all the NCCDS and DAS Scheduling Requests made by SWSI user
REQUEST_STATUS	Static	Contains description of all the valid status codes
REQUEST_TYPE	Static	
SAR	Dynamic	Stores information for NCCDS and DAS Scheduling Requests
SCHEDULE_CONNECTION	Static	Contains information about making connections with NCCDS scheduling system
SERVICE_LINK	Static	
SERVICE_PARAM	Static	Contains information for performing parameter validation, building dynamic display panels, and processing NCCDS GCMR/USM messages
SERVICE_TYPE	Static	Contains all the Service Types supported by SWSI and other information used for processing NCCDS GCMR/USM messages
SIC	Static	Contains all the SICs supported by SWSI
SP_ENUM_VALIDATION	Static	Contains information for validating parameters that of type Enumerated
SP_NUMERIC_VALIDATION	Static	Contains information about validating parameters that have valid range
SRM_RESULT_CODE	Static	Contains Text description of each SRM Result Code
SR_PARAM	Dynamic	Stores all parameters used in a SAR/RAR
SR_SERVICE	Dynamic	Stores service information for each event scheduled
SSC	Semi-Static	Contains SSC codes used to schedule SAR/RAR
SSC_PARAM	Semi-Static	Contains default values for each parameter assigned to an SSC code

SUPIDEN	Static	Contains valid SUPIDEN supported by SWSI
SWSI_USER	Semi-Static	Contains SWSI user information
SWSI_USER_SIC	Static	Intermediate table for SWSI_USER and SIC tables
TDRS_GROUP	Static	Contains TDRS group names
TDRS_IN_GROUP	Static	Intermediate table for TDR_NAME and TDRS_GROUP tables
TDRS_NAME	Static	Contains base TDRS names
UPD	Static	Contains information to build UPD display
UPD_ENUM_VALUE	Static	
UPD_LABEL	Static	
UPD_PARAM	Static	Contains information about UPD parameters to process NCCDS UPD messages and for constructing UPD display
USR_GCMR	Dynamic	Stores information for NCCDS User Reconfiguration Requests
ACTIVE_SCHEDULE_V	VIEW	
ACTIVE_SCH_SERVICE_V	VIEW	
SCHEDULE_REQUEST_V	VIEW	
SR_SERVICE_V	VIEW	

Abbreviations and Acronyms

AFN	Acquisition Failure Notice
ANCC	Auxiliary Network Control Center
API	Application Programming Interface
ASAR	Alternate Schedule Add Request
BB	Bit Block
CA	Certificate Authority
CCS	Communications and Control Segment
CM	Configuration Management
CVS	Concurrent Versions System
DAS	Demand Access System
DASCON	Demand Access System Controller
DCIR	Doppler Compensation Inhibit Request
DDD	Data Display Debugger
EIF	Engineering Interface
EIRP	effective isotropic radiated power
EUFUR	Expanded User Frequency Uncertainty Request
FDF	Flight Dynamics Facility
FLER	Forward Link EIRP Reconfiguration
FLSR	Forward Link Sweep Request
GCC	GNU C Compiler
GCM	Ground Control Message
GCMR	Ground Control Message Request
GCM S&D	Ground Control Message Status & Disposition
GDB	GNU Debugger
GDPro	Graphics Designer Professional

GN	Ground Network
GNU	recursive acronym for “GNU’s Not Unix”
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HA	High Availability
HP	Hewlett-Packard
HTML	Hypertext Markup Language
ICD	Interface Control Document
IDE	Integrated Development Environment
IIRV	Improved InterRange Vectors
IONET	IP Operational Network
IP	Internet Protocol
JSWITCH	Java-based Spacecraft Web Interface to Telemetry & Command Handling
JVM	Java Virtual Machine
KSA	Ku-band single access
KaSA	Ka-band single access
LDBP	Long Duration Balloon Project
MAF	Multiple Access Forward
MAR	Multiple Access Return
MGCMR	Multiple Ground Control Message Request
MOC	Mission Operations Center
MSOCC	Multisatellite Operations Control Center
PBKDR	Playback Deletet Request
PBKMR	Playback Modifucation Request
PBKR	Playback Request
NASA	National Aeronautics and Space Administration
NCC	Network Control Center
NCCDS	Network Control Center Data System

NCD	NCCDS Central Delogger
NISN	NASA Integrated Services Network
NPG	NCCDS Protocol Gateway
OOAD	object-oriented analysis & design
ODM	Operations Data Message
RADR	Resource Allocation Deletion Request
RAMR	Resource Allocation Modify Request
RAR	Resource Allocation Request
RCS	Revision Control System
RCTD	Return Channel Time Delay
RCTDM	Return Channel Time Delay Message
RFI	Radio Frequency Interference
RMA	Reliability/Maintainability/Availability
RR	Replace Request
SA	Single Access
SAA	South Atlantic Anomaly
SAR	Schedule Add Request
SDR	Schedule Delete Request
SIC	Spacecraft Identification Code
SMAF	S-band Multiple Access Forward
SMAR	S-band Multiple Access Return
SN	Space Network
SNIF	SWSI-NCCDS Interface
SRM	Schedule Result Message
SSL	Secure Socket Layer
SPSR	Service Planning Segment Replacement
SSA	S-band Single Access
SSC	Service Specification Code

STGT	Second TDRSS Ground Terminal
SUPIDEN	Support Identifier
SV	State Vector
SWSI	SN Web Services Interface
TBD	to be defined/determined
TBS	to be specified/supplied
TCP	Transmission Control Protocol
TDRS	Tracking and Data Relay Satellite
TLE	two-line element
TSW	TDRS Scheduling Window
TTM	Time Transfer Message
TUT	TDRS Unscheduled Time
UDP	User Datagram Protocol
UML	Unified Modeling Language
UPD	User Performance Data
UPDR	User Performance Data Request
UPS	User Planning System
URR	User Reacquisition Request
URRM	User Reconfiguration Request Message
USM	User Schedule Message
WLR	Wait List Request
WSGT	White Sands Ground Terminal
WWW	World Wide Web
XDR	eXternal Data representation
XML	eXtensible Markup Language